POLITECNICO DI MILANO
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

An Automatic Framework for Schema Mapping with Query Reverse Engineering

Supervisor: Prof. Letizia Tanca
Co-supervisor: Dr. Fabio Azzalini

M.Sc. Thesis by:
Gabriele Stucchi

Student ID:
928214

Academic Year 2020-2021

Abstract

Over the last few years more and more web-based information are characterizing our world. The major aspects of our life are described via data which are saved and stored in databases. These databases describe different domains and each one is composed by different data schemas. These schemas represent an event or an information of a specific domain, different schemas could depict the same data but in a different way. Implement more data integration systems is becoming a need since data are constantly evolving and schemas' volume is always increasing. Systems which applied to different schemas can provide a unified view of a specific domain that will give to the user a complete and reliable access to a domain. In our thesis we will propose a new framework for schema mapping, which is a critical step inside data integration. Schema mapping finds the relationships between the attributes of different schemas. Many of the existing frameworks used to create a schema mapping are thought for expert users who know how a database works and how to relate different schemas, that's why those frameworks are built expecting some inputs from the user that could help to reach the goal. With the expansion of the fields on which data are used, nowadays we see an increased use and need of data integration system also by unsophisticated users. These users don't know how a database works and for sure they will not be able to provide the inputs required by these frameworks. In this thesis we will present framework where schema mapping is achieved in a fully automatic way, preventing unsophisticated users to give inputs or to make reasoning on the results. Our frameworks uses the QRE algorithm, an algorithm which will return a set of queries that applied to a database will give as result the same tuples. Our framework has proven to be correct and complete on the datasets we have used since it returned the expected schema mapping.

Keywords: Data Integration, Schema Alignment, Schema Mapping, Query Reverse Engineering.

Sommario

Negli ultimi anni si è visto un incremento dell'utilizzo di informazioni digitali per descrivere il nostro mondo. I maggiori aspetti della nostra vita vengono descritti da dati che sono salvati su databases. Questi databases descrivono molte realtà e domini utilizzando molteplici schemi. Per un dominio possiamo avere diversi schemi che ne descrivono le informazioni ad esso legate, e questi schemi possono differire tra di loro ma allo stesso tempo rappresentare la stessa cosa. Considerando questo aspetto e il fatto che i dati continuano a cambiare e ad aumentare, è cresciuta la necessità di implementare sistemi in grado di unificare questi schemi per poter ottenere una visione unica di un certo dominio. Questa visione deve essere completa e affidabile per permettere ad un utente di poter accedere alle informazioni di un dominio. Nella nostra tesi presentiamo un framework utile ad ottenere uno schema mapping, uno degli aspetti fondamentali per poter trovare le relazioni tra i diversi schemi di uno stesso dominio. Molti dei sistemi utilizzati per schema mapping sono pensati per utenti esperti che sanno come interagire con un database, si aspettano degli input dall'utente per poter ottenere lo schema mapping. Considerando però l'incremento dell'utilizzo dei dati in ogni aspetto della nostra vita, è sempre più probabile che utenti meno esperti debbano interagire con i database e che richiedano quindi la necessità di utilizzare sistemi di schema mapping. Il nostro framework è pensato per essere completamente automatico, una volta ricevuti gli input iniziali ritornerà lo schema mapping finale senza dover chiedere ulteriori input all'utente, il quale potrà quindi essere sia esperto che meno. Per raggiungere il nostro scopo, nel nostro framework abbiamo utilizzato l'algoritmo di QRE, il quale ha come obbiettivo quello di ottenere un gruppo di query che applicate su un database ritornano gli stessi dati. Il nostro framework ha ottenuto come risultato lo schema mapping che ci aspettavamo venendo applicato sui datasets da noi usati, il quale era corretto e completo.

Parole chiave: Data Integration, Schema Alignment, Schema Mapping, Query Reverse Engineering.

Acknowledgements

Eccomi alla fine di questi sei lunghi anni di università dove ho imparato tanto e dove son cresciuto molto personalmente. Vorrei in quest'ultimà pagina dedicare qualche frase alle persone che mi hanno accompagnato in questo percorso.

Voglio iniziare ringraziando la Professoressa Letizia Tanca per avermi dato la possibilità di lavorare a questo progetto e ringrazio anche il Dr. Fabio Azzalini per avermi seguito pazientemente in questi mesi e per avermi sempre dato una mano quando ne avevo bisogno.

Voglio ringraziare principalmente mio papà e mia mamma che non mi hanno mai fatto mancare nulla e mi hanno sempre supportato. Mi hanno insegnato dei valori che spero di riuscire a trasmettere in futuro alla mia famiglia. Mi hanno permesso in questi anni di conoscere diverse realtà e che non sempre tutto è regalato. Un grazie non sarà mai abbastanza.

Un grazie anche a mia sorella Veronica che nel suo piccolo c'è sempre stata e la ringrazio soprattuto del fatto di essere una persona con cui in questi anni son sempre andato d'accordo. Ringrazio tutta la mia famiglia, dai nonni, agli zii e ai cugini che mi hanno sempre supportato in questi anni.

Un grazie speciale va a Francesca. Ormai da dieci anni una delle persone più importanti della mia vita. So di potermi fidare e soprattutto ho sempre potuto contare su di te. Grazie di tutto Cesca.

Ringrazio William e Sebastian che son gli amici di sempre, le persone con cui ho trascorso più tempo e esperienze in questi anni e che ci son sempre stati per me. Alla prossima birra o alla prossima partita di basket o calcetto, Grazie.

Ringrazio i miei amici del liceo che mi son sempre stati di sostegno in questi anni e hanno creduto in me.

Ringrazio i miei amici di università con cui ho affrontato questo lungo percorso e durante il quale hanno sempre cercato di stimolarmi in meglio.

Ringrazio Gregory e Marco, i miei primi amici. Conosciuti alle elementari e a cui son sempre stato vicino anche se durante questi anni abbiamo intrapreso strade diverse.

Ringrazio infine tutti gli altri miei amici e le persone che ho conosciuto in questi anni e che mi hanno permesso di crescere e diventare la persona che sono oggi.

Grazie per questi anni indimenticabili.

Gabriele Stucchi
7 Ottobre 2021

Contents

List of Figures

Chapter 1

Introduction

1.1 Context

Over the past years the amount of structured information exploded and as a result many users, also non-technical users, have to deal with tasks that requires to combine, structure and work with different schemas. These schemas are data structures, called database, that have tables on which they store information. These tables are formed by rows and columns. The first ones are called tuples and they are an aggregation of values which represent a specific information (which could be an event or for example the records of a person). The latter one are also called attributes and with the values presented in the tuples specifies what these values are referring to.

In these year we witnessed an explosion of the dimension of web-based datasources, in the size of the data but also in the numbers of schemas that represent a domain. This increase happened very fast, thus the data are dynamic information that could change rapidly. This change could also bring to different variety between the schemas, we could have different schemas that represent the same domain but they are totally distinct between them. These factors create also schemas with different qualities, some of these schema could be inaccurate or not fully covering the respective domain. For these reasons the need for data integration system is risen. The need to align different schemas became significant and with this, also the necessity to link the attributes of the schemas in the same domain to obtain a unified view which will give a complete and accurate vision of the specific domain.

The use of **Schema Mappings** method increased and several different systems have been developed, each one with their characteristics. Schema mapping methods are the fulcrum of data integration systems, they are the last step in the schema alignment procedure. They specify the semantic relationships between the attributes of different schemas. The majority of these Schema mapping systems provide a framework which, with the help of input taken by the user, create a mapping between two different schemas. For modern schema mapping tasks this is not suitable, it will require that the user knows in details both the source and target schemas. As we already said in these years the growth of the web-based information brought unsophisticated users to deal with this problems and these users don't have knowledge of the structures of the schemas or how to create attribute-level matches. A sentence taken from [12] points out this problem:

> *Non-technical users should be able to cook their data with their own flavor, even if they cannot master the "professional kitchenware" designed for database experts*

1.2 Proposed Solution

With this increased need of schema mapping methods usable also by unsophisticated users, I developed a fully automatic framework that given in input the source and the target schema, and a result table, obtains a schema mapping without asking supplementary inputs or information to the user. Our framework is based on the use of the **Query reverse engeneering (QRE)** algorithm. This algorithm given a dataset and a result table gives as result one or more SQL queries, we will call them IEQs, that applied to the dataset would return the tuples in the result table.

The aim of this thesis is to find a schema mapping using as first step the QRE algorithm modified according to our needs. We made the QRE able to find queries also for categorical attributes (string encoding) and then we also modified it in order to obtain not only a query but a pre-determined number of queries. Then with a comparison between the IEQs obtained from the source and from the target we will create a mapping between the source and the target based on the attributes in the SELECT and WHERE clauses of these queries. Our results will be saved in an external matrix which will give a visual representation of the generated mapping.

1.3 Thesis Structure

The thesis has the following structure:

- ☐ In **Chapter 2 - Background** are introduced the functions necessary to create the datasets and the function used to compare the IEQs. Then are introduced also the theoretical concepts needed to understand the discussed topics.
- ☐ In **Chapter 3 - State of the Art** there is the detailed explanation of what is schema mapping and it is provided an overview of different studies applied in the same field. At last there is an overview of the QRE algorithm, how it works and its functionalities.
- ☐ In **Chapter 4 - Methodology** are explained the steps of our process and the reasons behind our methodological choices. We provide also a use case that helps to understand how it works.
- ☐ In **Chapter 5 - Datasets and Experiments** are introduced the Datasets used for our studies and how we have obtained them. We also have provided some tests to better clarify how the methodology work.
- ☐ In **Chapter 6 - Conclusions and Future works** are synthetized the results obtained from the thesis and lastly are suggested possible future works that could improve and optimize our studies.

Chapter 2

Background

In this chapter we are going to provide some backgrounds necessary to understand the topics discussed in the next chapters. We initially introduce the Data Integration problem, the field on which our thesis is working. Then we will introduce some functions that will be useful in our methodology.

2.1 Definitions

In this section we will add some definition that will come in handy going forward.

Definition (SPJ Query). A Select-Project-Join (SPJ) query Q, is a SQL query containing only three clauses: select, from and where-clause. The select-clause is where the desired attributes to project are specified, the from-clause specifies the tables from which those attributes come from and the where-clause is where the predicates for selecting those attributes are specified.

Definition (Instance-Equivalent Queries (IEQs)). Let Q and Q' be queries, Q and Q' are instance equivalent queries if both produce the same output w.r.t some database D, i.e., $Q(D) \equiv Q'(D)$.

Definition (Datafication). The process that transform each event, subjects, objects and every interaction in the word into digital data.

2.2 Data Integration

Since in the last years we have seen, and we cite [15], a rise of digital technologies, digitalization and big data, we are witnessing an intensification of datafication and a successive need to address the big data integration challenge. Data integration increase is in fact associated with the explosion of the volume of data and the need to share these data, possibly coming from different source, with a unified view.

The big data integration challenge is characterized by the "V" dimensions which are the dimensions of data sources.

- **Volume**: with this dimension we are referring not only to the size of data in a schema, but also to the increased number of schemas, even for a single domain. In a study from Dalvi et al. [6] they found out that the domains they have considered were composed by thousands to tens of thousands of web sources, while traditional data integration data sources doesn't reach these types of numbers.

- **Velocity**: Since we have an increase of datafication and so of available data, the number of data sources is rapidly exploding and many of them are quite dynamic. Information changes over time, we need to provide an integrated view of this changing data. With a study of He et al. [10] in 2004 there is an estimate of 1,258,000 distinct query interfaces to deep web content. After two years, in 2006, a study by Madhavan et al. [13] estimates 10 million distinct query interfaces to deep web content. This is a reflection of the velocity of how the number of deep web sources increased.

- **Variety**: We have a huge variety of data sources from different domains. Moreover data sources for similar entities can present variety in how they structure their data or how they describe an entity. Lastly, data sources change over time creating more variety that need to be handled. In the studies of Li et al. [11], they have identified a high variety of the sources in the Stock domain. Another study, by He et al. [10], identified that deeb web databases have a high variety, classifying 51% domain in non e-commerce and the remaining in e-commerce.

- **Veracity**: We can see that data sources have different quality, they are dissimilar in accuracy, coverage and timeliness of data provided. This dimension is becoming a considerably problem since the number and diversity of data sources increased. With this problem is clear that we cannot rely only on one source of a domain but we have to take a look also on the other ones.

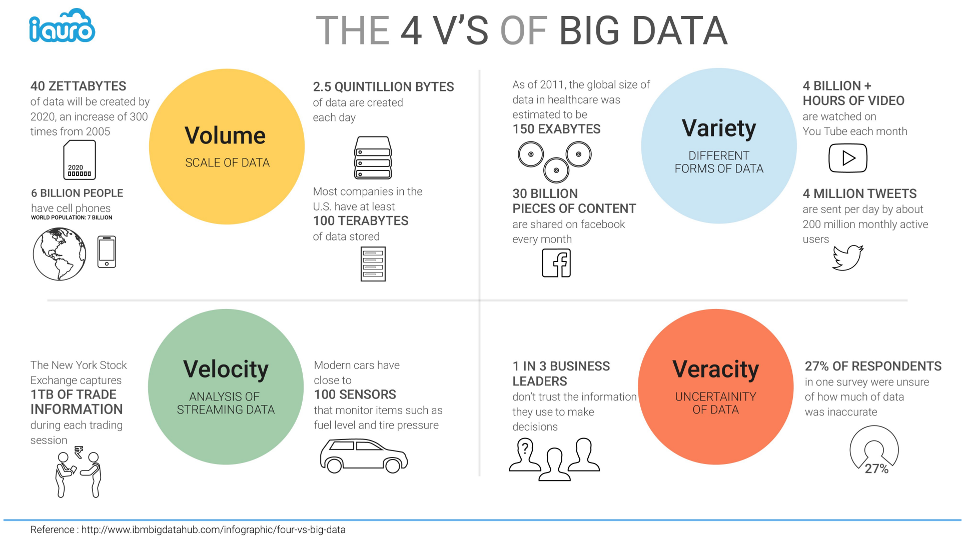

THE 4 V'S OF BIG DATA
Volume
SCALE OF DATA
40 ZETTABYTES of data will be created by 2020, an increase of 300 times from 2005
6 BILLION PEOPLE have cell phones
WORLD POPULATION: 7 BILLION
2.5 QUINTILLION BYTES of data are created each day
Most companies in the U.S. have at least 100 TERABYTES of data stored
Variety
DIFFERENT FORMS OF DATA
As of 2011, the global size of data in healthcare was estimated to be 150 EXABYTES
30 BILLION PIECES OF CONTENT are shared on facebook every month
4 BILLION + HOURS OF VIDEO are watched on You Tube each month
4 MILLION TWEETS are sent per day by about 200 million monthly active users
Velocity
ANALYSIS OF STREAMING DATA
The New York Stock Exchange captures 1TB OF TRADE INFORMATION during each trading session
Modern cars have close to 100 SENSORS that monitor items such as fuel level and tire pressure
Veracity
UNCERTAINITY OF DATA
1 IN 3 BUSINESS LEADERS don't trust the information they use to make decisions
27% OF RESPONDENTS in one survey were unsure of how much of data was inaccurate
27%
Reference : http://www.ibmbigdatahub.com/infographic/four-vs-big-data

Figure 2.1: Dimensions of Data

Data integration consists of three major steps:

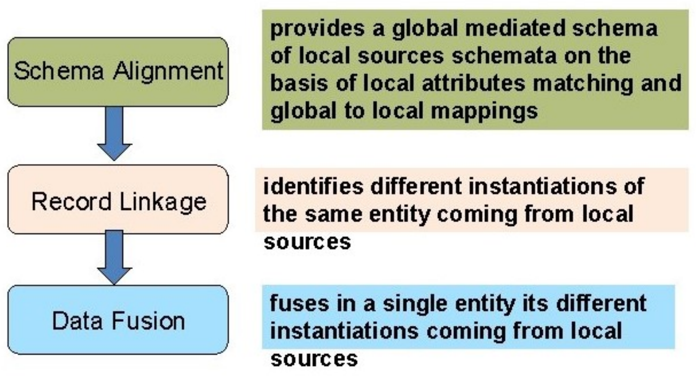

Schema Alignment
provides a global mediated schema of local sources schemata on the basis of local attributes matching and global to local mappings
Record Linkage
identifies different instantiations of the same entity coming from local sources
Data Fusion
fuses in a single entity its different instantiations coming from local sources

Figure 2.2: Steps of Data Integration

2.2.1 Schema Alignment

The first one, Schema Alignment, has the goal to link the attributes that have the same meaning between different sources and it also address the problem of **semantic ambiguity**. The latter is the fact that in different sources the same conceptual information could be presented differently but also that different conceptual information could be presented similarly. As we can see in [9], given the grew of available online ontologies which represents different domains with a lot and diverse point of view, we have faced an increase number of ambiguity problems.

Passing from integrate data in an organization to integrate data in a big data environment made the problem tougher, since we are now dealing with an increased number of data sources. This volume and velocity increased also the variety of data, requiring new techniques to solve schema heterogeneity.

Schema alignment contains three classic steps:

- **Mediated schema**. A unified view of the different sources, it usually contains more information than each schema but in some cases may not present all the information.

- **Attribute matching**. Attributes in the mediated schema are matched with the attributes in source schema.

- **Schema mapping**. A mapping that specify semantic relationships among the attributes of different data sources.

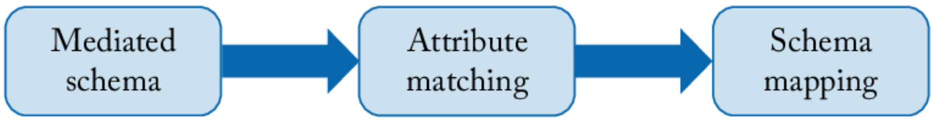


Figure 2.3: Steps of Schema Alignment

Schema mapping is the fulcrum of a data integration system but it is not easy to create and maintain those mappings since we have a huge number of data sources with increased variety. A solution for this variety and so uncertainty on how to model the domain, is to create probabilistic schema mapping which contains a set of attribute, each with a possible matching between source attributes and the mediated schema's one. So we will have different mappings that could be the correct and we will have to make a study to understand which one is the correct one.

Now that we have a general knowledge of what is schema alignment, and introduced the main concept of our thesis which is Schema mapping, a closer look in Chapter 3, we will explain the next two steps of Data Integration. We will not go in details about these ones but we will only give a general idea.

2.2.2 Record Linkage

Once we have aligned different schemas, we still could have that values provided by the sources for the same attribute may differ (naming conventions, typo, and so on). With **Record Linkage** the objective is to assign each record to an entity and so understand which records refer to the same entity. We have a lot of records to be linked and this number become even greater in the big data environment demanding for appropriate record linkage techniques. So, the basic record linkage, besides its basic steps, has to implent specific techniques to handle the challenges given by the dimensions of the big data word.

One of the basic steps is **pairwise matching**, it compares different records and determine if they refer or not to the same entity. It could be done with different approaches:

- **Rule-based**: it applies domain knowledge to create matching rules that bring to a decision. It could deal with complex scenarios but requires a lot of domain and data knowledge.
- **Classification-based**: it trains a classifier using positive and negative examples, then the classifier classifies a pair of records. It doesn't require domain knowledge but it needs a large number of training examples.
- **Distance-based**: it computes the similarity between the attributes values using string comparison techniques (described in the previous section). It sets thresholds that indicates if there is a match, a possible match or no-match. Domain knowledge is limited to state distance metrics on atomic attributes.

Within this step there could be inconsistency between the matching records, we could have for example that a pair of records R1 and R2 matches, than R2 matches with R3 but R1 does not match R3. To reach a globally consistent decision on how to classify all records, we have the **Clustering** step. First, with pairwise matching we create an undirected edge between the matching records, than different partitions are created. Those partitions are such that every partition does not have exiting edges conneting with other partitions. Each partition is what is called a Cluster. So we have a graph clustered in different partitions that are disjoint.

These two techniques could be quite inefficient and infeasible for set with a large number of records. So, it was proposed a new strategy, called **Blocking**, that has the goal to partition the input into different small blocks and than on these blocks it applies pairwise matching.

As we said before, Record Linkage become a challenge when it is applied to big data sets. So, it has been studied complementary techniques that address this problem, specific to each dimension. We will not go in further details about each techniques because they are not topics of our thesis, we will only list them:

- **Volume**: Two major techniques, one uses MapReduce which are effective in parallelizing data, the other one uses multiple blocking functions and identifies the most promising pairs of record.

- **Velocity**: Since many of the sources are dynamic, it is necessary to perform an incremental record linkage, which will update existing linkage results upon the arrival of new data. There are already some clusters, new records are put into existing ones or in a new cluster.

- **Variety**: Even if we have already had the Alignment step, with the amount of domains and sources in the big data era we need this further step. It will create more links between attributes name and structured records.

- **Veracity**: Tries to create clusters that identifies records which refer to the same entity but with different values or in different points in time.

2.2.3 Data Fusion

The last step of Data integration is **Data fusion**. Different sources could provide information for the same attribute of the same entity that may have conflicting values, Data fusion seeks to understand which value truly represents the entity. As for the previous steps, the increase volume of sources increased the number of conflicting data. Data fusion decides the value, or the set of values, or the list of values that is consisted with the real world. At the beginning, before the big data era, the approaches were often rule based, which became inadequate for higher volume of data where there is large veracity. To address this problem, many techniques tried to identify trustworthy sources and detecting copying between the sources. In these techniques we can have some of all of the following aspects:

- **Truth discovery**: find the true value between the conflicting ones.

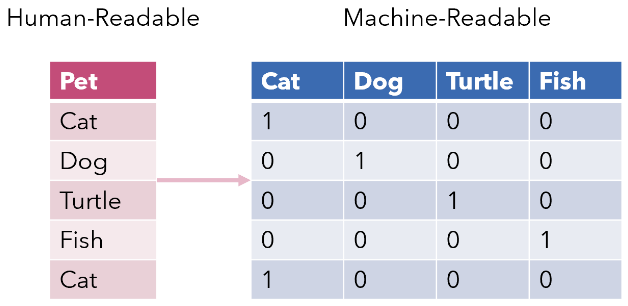
- **Trustworthiness evaluation**: Evaluate the trustworthiness of a source according to the correctness of its values.

- **Copy detection**: Detects copying between data sources.

The second aspect is one of the most important point of data fusion. It is calculated as the accuracy of a source, A(S), which is the probability that a record in the source S is true.

2.3 Encoding

In this paragraph we are going to explain what is Encoding and how it can be done. First of all, why we are interested in Encoding? The QRE algorithm (introduced and explained in Chapter 3) that we are going to use, works only on numerical datasets. Our thesis provides a system to perform schema mapping between two datasets that can present some categorical attributes. Given that our approach is based on the QRE algorithm Talos presented in [14], an algorithm that as we just said works only numerical data, we need to encode our datasets before we can use them. Beside that, encoding is very useful and required in most of the machine learning algorithms. We have used the one hot encoding approach, a process that converts categorical data variables into numerical data. How does it work? One hot encoding takes the categorical variable and for every value present in the column of this variable create a new column assigning a binary value of 1 or 0 to those columns. Here an example for a better understanding:



Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Figure 2.4: Example of One Hot encoding

Why we have used one hot encoding? There are some encoding algorithms, like label encoder, that does not create new columns but converts the values in the original columns with integers. At the end we will have the original column with the categorical value replaced by numerical values. We need the originals strings to perform our search so using label encoder we would lose the information. Generating these numbers, label encoders create also an order in the column that for some categorical values (like cities) does not have sense. Some machine learning algorithm could use the order of numbers as an attribute of significance. This could be helpful in some cases but in ours is not so we preferred to use one hot encoding.

With one hot encoding we will create new columns for every value that we have and we can place as header of the column the actual value of the categorical attribute so we do not lose it. We do not want to lose it because we will need it also to find the final query, it could be part of a where-clause. Now that we have decided to use one-hot we have to choose between one hot encoding with Pandas or with Sklearn since our system is implemented with python.

- **Pandas:** In Pandas we have the function pandas.get_dummies() that given a categorical dataset and possibly some other parameters, will return the original dataset converted in a numerical dataset with dummy variables. Those dummy variables are new columns of the dataset that are created from the values present in the categorical columns. As see in Fig. 2.4 from one column whit three different values we obtain three numerical columns.

- **Sklearn:** In Sklearn we have the function OneHotEncoder that encode categorical features as a one-hot numeric array. We give as input a categorical column of our dataset and the function will return a matrix composed by bynary column for each category. The categories are derived based on the unique values in each feature. The difference with get_dummies() is that OneHotEncoder creates a function which persists so it can be applied to new data using the same categorical variables.

For our thesis we have chosen get_dummies() because we have seen that we obtain the same results in both cases and we don't take advantage of the characteristics of sklearn encoder. Furthermore, when we have tried OneHotEncoder from sklearn we had to built an external function that first needed to find all the categorical columns. Then, on those columns, we needed to call the encoder and we obtained a dataframe for each column. Once we had those dataframe we merged them to obtain our final dataframe with all the categorical columns encoded. Instead, with pandas encoding we only had to call the function to obtain the same result.

2.4 String Comparison

In this section we are going to explain what is string comparison and why we need it in our thesis. In Chapter 4 we will see how it is used in our methodology.

In the easiest case we are comparing two strings that are exactly the same and so the comparison is trivial. For our work we cannot stop our reasoning on the equality of the strings but we must take a look also at the similarity that they have.

An example of why it is important to look at the similarity: Suppose you need some user input. It could happen that there will be a typo in the input. To bypass the typo there exists string matching techniques that compare the input with a given string and return the percentage of similarity between the strings.

We can have different of algorithms:

- **Edit distance based**: these ones calculates the number of operations necessary to modify one string into another. When we have less edit operation it means the strings are similar.
- **Token-based**. Here we have algorithms that compare the set of characters in both strings and we have a higher similarity when we have a high number of common characters.
- **Sequence-based**. Lastly, in this category the algorithms compare sub-strings, trying to find the longest sequence in common. Higher the sequence, higher the similarity.

The first techniques that we are going to explore is the **Levenshtein distance**, an Edit distance based algorithm. Used mainly to address typos, it calculates the numbers of transformations needed to transform a string into another one. Here we can see the definition of the Levenshtein distance:

From this definition is calculated a ratio which is used in the **Fuzzy-Wuzzy Package**, the ratio is obtained from this formula: $\frac{(|a|+|b|)-lev_{a,b}(i,j)}{|a|+|b|}$

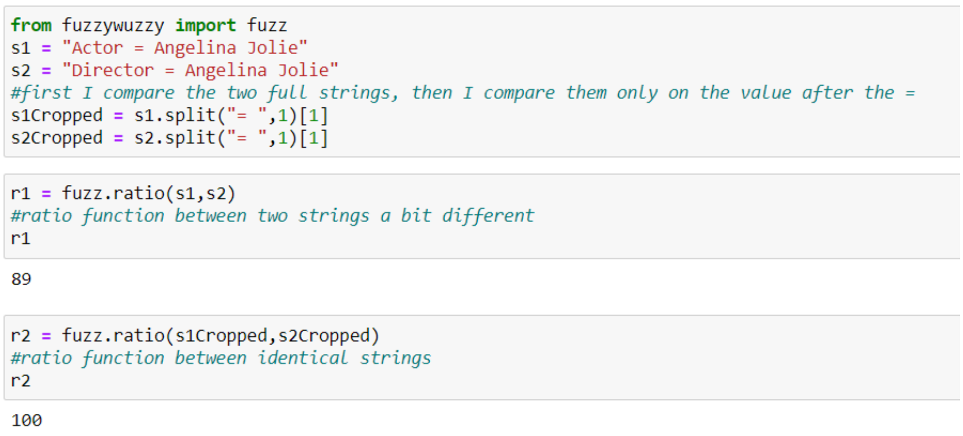
This package provide different functions that handles cases similar to the previous one but where the Levenshtein distance would fail. In the simplest case we can use **fuzz.ratio(s1,s2)** (where s1 and s2 are two generic strings) that computes the standard Levenshtein distance similarity ratio. It will return a float in the range [0,1]. As we said, this package has other functions that comes in handy in different situations. For example we could check if a string is a substring of another string using **fuzz.partial_ratio(s1,s2)**.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

Figure 2.5: Levenshtein Distance

The algorithm compare the substring s1 with length k with all the possible substrings in s2 that have length k and return the best score of the comparison. Another important function is the **fuzz.token_sort_ratio(s1,s2)** that order alphabetically the strings and then applies fuzz.ratio to obtain the similarity percentage.

We could get lost with other functions present in this package but for our thesis we will need to compare two strings that could be the title of a film or the name of an actor and so on. So we will not need complex functions but we can simply use the fuzz.ratio function (same of Levenshtein distance as we said). Here a practical example:



```
from fuzzywuzzy import fuzz
s1 = "Actor = Angelina Jolie"
s2 = "Director = Angelina Jolie"
#first I compare the two full strings, then I compare them only on the value after the =
s1Cropped = s1.split("= ",1)[1]
s2Cropped = s2.split("= ",1)[1]
```

```
r1 = fuzz.ratio(s1,s2)
#ratio function between two strings a bit different
r1
```

```
89
```

```
r2 = fuzz.ratio(s1Cropped,s2Cropped)
#ratio function between identical strings
r2
```

```
100
```

Figure 2.6: Example of Fuzz Ratio

We can have different functions not based on the Levenshtein distance, here we will propose some different solutions that could have been used in our thesis. Firstly we have the **Jaro Similarity**, also an edit one, that will return a value in the range from 0 to 1. The formula used by this function is:

With m representing the matching characters and t is half the number

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Figure 2.7: Jaro's Formula

of matching characters in different order. $\mid s1 \mid$ and $\mid s2 \mid$ are the lengths of the strings. Like in fuzzy we have a more sophisticated jaro functions, the **Jaro-Winkler Similarity** which adds to the first one a more accurate answer when two strings have similar prefix.

The previously presented methods provided a coefficient which indicated the similarity between the strings, the following one returns a value that indicates how dissimilar two strings are. We are talking about the **Jaccard Distance**, a token-based algorithm. It still range from 0 to 1 but we have that the strings are more similar when the value is low. The Jaccard Distance is calculated as follows: D(X,Y) = 1 - J(X,Y). Where the last term is the Jaccard Similarity Index which is equal to $|X \cap Y|/|X \cup Y|$, the characters in both sets and the total number of characters. Another token-based algorithm is the **Sorensen-Dice** one. It differs from the Jaccard Distance algorithm because it counts twice the characters that intersect in the strings and plus at the denominators we don't have the union of the sets (strings) but the sum of all characters in both strings. This may bring to a overestimate of the similarity.

We close this section with a final example of a sequence based algorithm. We presents the **Ratcliff-Obershelp** algorithm. It starts by finding the longest common substrings, split the strings on it and tires to find other substrings in the splitted strings. It goes on recursively until a splitted string has a length lower than a default value. The result of the similarity is calculated as twice the sum of common characters divided the total number of characters.

2.5 Sklearn Decision Tree Classifier

In this paragraph we are going to discuss what are Decision Trees and how they are used in the QRE algorithm. A decision tree is a non-parametric supervised learning method used for both classification and regression. It is based on a tree structure where an internal node is an attribute on which there will be a test that will create a branch, and each leaf nod represents the outcome. A leaf node represents a class label. At each node an attribute is taken to split examples into distinct classes as much as possible. Taking different branches in a decision tree means to select the data that have specific characteristic, in fact every branch is obtained by applying a condition to an attribute and more deeply we go in a decision tree more condition are applied at each node. The path from root to leaf, or to each node, represent classification rules.

But how does a decision tree chooses the attribute on which to split? It selects the best attribute using Attribute Selection Measures. This is a heuristic for selecting the best attribute to split. Most popular selection measures are Information Gain, Gain Ratio and Gini Index. Depending on the measure used to split we can have different types of decision tree algorithm, respectively: ID3, C4.5 (an improvement of ID3) and CART (Classification and Regression Tree). We will focus on the last one which is the one used by our sklearn Decision Tree.

This decision tree algorithm split the attributes looking at the Gini Index:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Figure 2.8: Gini Index

Where Pi is the probability that a tuple in the dataset belongs to class Ci. The algorithm is going to split on the attribute that has the lowest Gini Index given a possible partition of the data set D into two data set D1 and D2. The weighted sum of the impurity of each partition is given by this formula:

We have made an optimal split but we aren't sure that this will lead to optimal splits in following nodes, this is the reason why decision tree

$$\mathrm{Gini}_A(D)=\frac{|D1|}{|D|}\mathrm{Gini}(D_1)+\frac{|D2|}{|D|}\mathrm{Gini}(D_2)$$

Figure 2.9: Gini Index after split on attribute A

algorithm are **Greedy** algorithm. The next figure, 2.10, shows how the splits works. In the first node we split on the attribute sex and we obtain two nodes where we can see that the next split will be on different attributes (Age and Pclass) given the Gini Index:

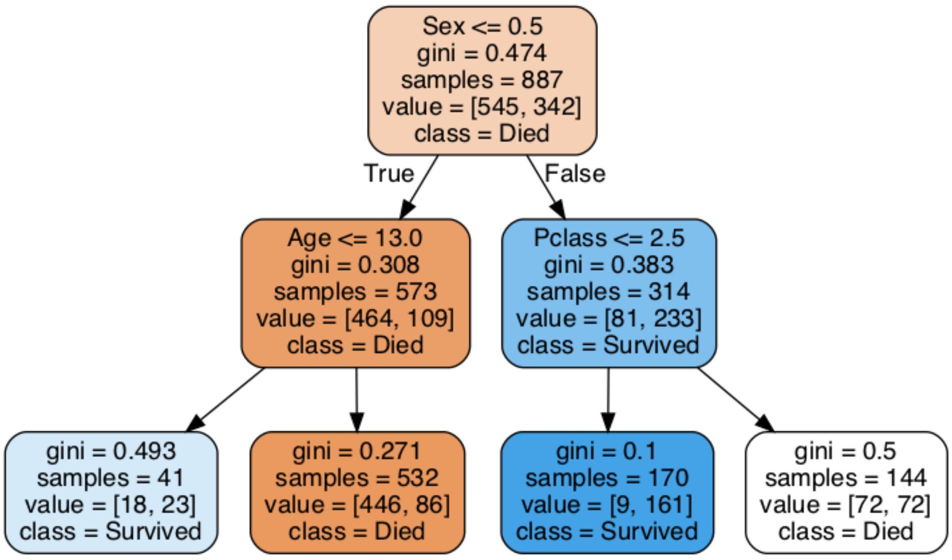


Figure 2.10: Example of Decision Tree

Chapter 3

State of the Art

In this chapter, we will go over some of the papers used as a base for our thesis. First, we will introduce the concept of schema mapping presenting other methods along with their limits. Next, we will focus on the paper that describes the QRE algorithm that we have used in our thesis.

3.1 Schema Mapping

> *A schema mapping transforms a source database instance into an instance that obeys a target schema. It has long been one of the most important, yet difficult, problems in the areas of data exchange and data integration.*

With this piece taken from the introduction of the paper [12] we are introducing the problem of schema mapping that is what we want to accomplish with our thesis. What is schema mapping? Schema mapping is a task that find the relationships between schemas. Given a source schema S and a target schema T we can find a schema mapping M that correlates the attributes of the different schemas. We can define a schema mapping also as we see in Bonifati's paper [4]: a schema mapping is a combination of declarative specification of the semantic relationship between elements of a source schema and a target schema. Schema mappings is also used for data exchange with the objective to create an instance that reflects the source instance.

Schema mappings can be distinguished in three types:

- **GAV**: global-as-view, which obtain the data in the mediated schema querying the ones in source schemas.

- **LAV**: local-as-view, the mediated schema is used to provide a view of the source data.

- **GLAV**: global-local-as-view, we have a virtual schema that provides a view both of the mediated and local schema.

In our thesis we will apply a new schema mapping technique to a source schema and to a target schema with the goal to find the attributes in the source schema that represents the same attributes of the target schema. Our technique will be completely automatic upon the input of one example schema.

Before presenting our works we will focus on different systems that create mappings and that have an interaction with the user.

Clio, [7], was one of the first Semi-automatic tools created to do schema alignment. Clio is one of the earliest project to use schema mappings with the goal to simplify information integration and to find the relationships between data in heterogeneous schemas. Clio had these requirements:

- We do not assume any relationship between the schemas. We can have schemas with their data and constraints.

- We need to be able to map between relational and nested schemas.

- We must be capable to generate mappings with diverse levels of granularity.

- The algorithm that creates the mapping has to be incremental. Incomplete mappings could provide sufficient information to the user and they could be refined over time.

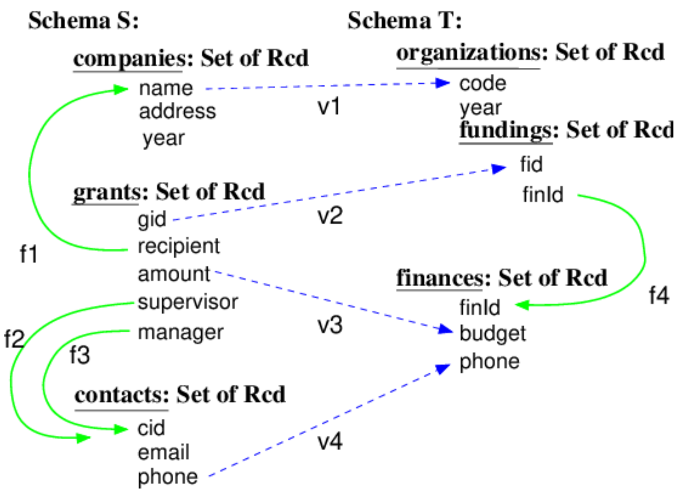


Figure 3.1: Possible mapping between a Source and a Target schema

Clio's project assumes that we want to describe a mapping between two schema, called source and target. Suppose we have these two schemas, when Clio is applied to them, it will create a possible combination of mappings if there are clear associations between target and source. The user is provided with a data viewer which allows him to see the mapping and decide which one to use.

Two principal systems that allow to non expert users to create schema mapping are:

- **IMF**: Interactive Mapping Specification with Exemplar Tuples" by Bonifati.
- **MWeaver**: Sample-Driven Schema Mapping by Jagadish

The latter one, [12], proposed a system called MWeaver, that provide mappings with the help of the user that provide data in a spreadsheet-style interface. The ground basis for this system is that with the increasing amount of data, non-technical user should be provided with a system that allows them to make data integration. In this paper they proposed a sample-driven approach that allows those non-technical user to construct their own data. The system finds the possible mappings and with the information given by the user, it can conclude which are the best mappings. It is user friendly because the user doesn't need to understand the schema of the source or the mappings and the user has only to provide some information as input. Two challenges faced by this system are the facts that it has to create mappings only from the user-provided samples, and than that it has to compute the mappings quickly to provide a feedback to the user and have a response. As results they have obtained a system that is user friendly, they developed an efficient sample search algorithm, and they provided a quite efficient system to obtain a schema mapping. Differently from other system, like [7], that creates mappings and then ask user to debug them, MWeaver require the user to trust the mappings and ask only for information. The only feedback the user can give are additional data which will help the system to understand which is or are the correct mappings currently generated. The limit of this work is that it requires iteration with the user and if the source database has ambiguous attributes the system could necessitate of a large number of samples from the user.

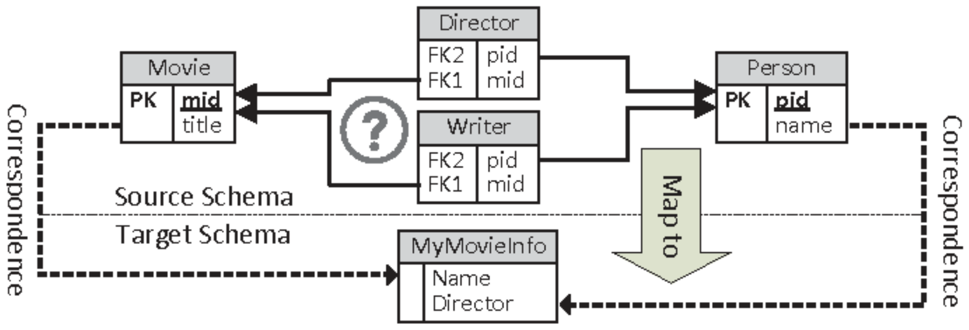


Figure 3.2: Example of Mapping from Jagadidsh

Now we will focus on the first one, [4], an interactive framework for non-expert users that creates schema mappings. The idea is to create exemplar tuples that refers to some schema mappings and with an interaction of the user under the form of boolean queries, we can test the validity of these tuples. This approach, Interactive Mapping Specification (IMS), with a set of exemplar tuples provided by non-expert users, pose to the user simple boolean questions with the intent to derive the correct mapping. Those exemplar tuples could have ambiguities or could be ill-posed that could lead to create mappings too specific. Keeping this in mind, the framework will derives smaller and normalized mappings, closer to what the user wants. This work is limited by the possible ambiguity present in the tuples and the fact that the mappings are created only from the attributes present on the tuples. We could have sources with many attributes and we can not pretend to ask the user a certain number of tuples to handle all the relationship. We can also see that this framework depends a lot on interaction with users.

Important studies on mappings have been done by **Atzeni**, so now we will introduce one of his papers, [2], that focus the problem to reuse previously defined schema mappings and also use them to create new mappings.

The creation of mappings is a time-consuming task and the objective of the framework from Atzeni is to address this problem by creating meta-mappings that give an opportunity to reuse some mappings. The challenges are that they needed to understand what is a generic mapping for an original schema mapping, to find a mechanism to generate and store all the possible combination of constraints for a mapping and lastly to create tools that helped them to choose a generic mapping for new schemas. With GAIA, the system presented in this paper, they reached the goals to deduce generic mapping from input schema and also to have a ranked list of possible generic mappings for an input source and target schema. They concluded that creating a repository with all the created schema mappings will provide a suitable schema mapping to a new scenario in high efficient time respects with a classic schema mapping method.

Those systems took care of how to create a schema mapping between two schemas but those mappings could be complex and so increased the need to create tools to understand them. In the paper [1] of **Alexe**, it is studied an approach that given schema mappings provide data examples which illustrate these schema mappings. They search for data examples that are a unique characterization of a schema mapping, they consider finite sets of data examples that are positive or negative examples which will characterize the schema mapping. Unfortunately they found out that that some schema mappings are not uniquely characterizable by those sets. So, it has been introduced the concept of universal example and universal solution, the most general possible solution and represent the entire space of solutions for a source. They concluded that the first tipe of data examples, the positive and negative one, are not sufficient and do not yield to unique characterization of schema mappings. With universal example they overcome some of the issues they encountered but still with some limitation.

We have introduced three schema mappings systems and now we will introduce the paper of Yan, [3], that present a solution to evaluate those mappings. Firstly, it introduced the concept of **benchmarks**: we need something to evaluate the mappings, an evaluation scenarios. Designing these is not easy, because we have not only one correct set of mappings and we have not a clear specification of the input language. Different tools could provide a different solution but it does not mean that the generated mapping is wrong, simply we have a lack of agreement on the semantics of the matches and some tools may interpret differently the sources.

3.2 Query Reverse Engineering

In previous chapters we have introduced the concept of data integration and precisely the step of schema mapping in schema alignment. Our thesis want to present a new schema mapping method based on the **Query Reverse Engineering (QRE)** algorithm introduced in the paper of Tran et al [17].

Definition: Given a database D and a result table T (output of some known or unknown query Q on D) the goal of QRE is to reverse-engineer a query Q' such that the output of query Q' on database D is equal to T.

This is the formal definition on QRE, and as we can see the goal is to find a query that produce the same output of another query, these two queries are called instance-equivalent queries (IEQs). They are IEQs if they produce the same results w.r.t a database D.

What is the purpose of QRE? We can have different use cases of QRE, here we will propose three of them to better understand the motive of QRE:

- **Database usability:** With the use of QRE, a user can derive IEQs of a given query on a database and these queries can provide to the user different characterizations of the tuples obtained from the first query. IEQs can uncover hidden relationships among the data which could help in better understand complex database. This could also provide information useful to the user to create more specific queries.

- **Data exploration & analysis:** In this case we don't have the input query and QRE help us to have IEQs derived from the input database and the input result. These queries will describe the possible characteristics of the tuples in the result. This scenario could easily happen for different reasons (change of the software, missing or inaccurate documentation, etc.)

- **Database security:** An attacker could target with different queries the same set of tuples and obtain sensitive information. With QRE, having IEQs generated for historical queries, this could be prevented because the IEQs will understand that the different queries are targeting the same tuples and that this could violate the privacy constraints.

In the Tran et Al paper they provided a QRE algorithm which is an extension of the query by output (QBO) problem, which is a simplified version of the QRE. With their algorithm they now provide the possibility to apply the algorithm not knowing the original query Q, the IEQs derived are more expressive and not only SPJ, finally they considered the fact that the database D could change and so have multiple versions.

In the paper they introduced the **TALOS** approach that resolve the QRE algorithm. To create the IEQs we need to determine the three components of a SPJ query:

- **rel(Q'):** relation in the from-clause;
- **sel(Q'):** conditions in the where-clause;
- **proj(Q'):** attributes in the select-clause;

The rel(Q') component is a subgraph of the schema graph and this provide alternative characterizations if Q(D) which involve different join paths. The critical part is the sel(Q') component,because it has to be "minimal" (without too many condition) and insightful. In the paper they have handled the sel(Q') part as a classification problem, they have classified the tuples in a subgraph as positive if they contribute to the query result or negative if they don't, and the sel(Q') is obtained by the conditions that selects the positive tuples. The proj(Q') is obtained by the attributes in the select-clause of the input query, or if the query is unknown, they are derived.

As classifier they have choosen decision trees, as a form of rule-based classifier. We have explained how a decision tree works in Chapter 2, for QRE they need to be adapted and they have faced two key challenges:

- **At-least-one semantics:** Multiple tuples could be projected to the same tuple and so it is difficult to assign a correct class to each tuple. There could be tuples labeled as negative that have the same projection of positive tuples. We can have BOUND TUPLES or FREE TUPLES. First ones must be negative or positive either they are in the output set or they are the only tuple in some subset. The latter are contained in subsets with other tuples and could be labeled positive or negative. Here the classification is more flexible in the class label assignment, at least one tuple from each subset must be labeled positive.
- **Performance issues:** how to efficiently generate candidates for rel(Q') and optimize the computation of the table required for the classification step. TALOS exploits join indices.

With at-least-one semantics they had to study a new approach to calculate the optimal node split given the presence of free tuples, how to compute the optimal Gini index without enumerating all the possible labels of the tuples. To this scope they have thought about five combinations on how to consider the free tuples. Our thesis is based on the first one which maximize the number of positive tuples in both splits, so all the free tuples are converted to positive one.

Now that they have discussed how to classify the tuples, they introduced the decision tree used in their project. They have chosen to adapt an existing decision tree classifier, SLIQ, which is an efficient decision tree classifier designed for handling large training data. SLIQ creates an array called class list which is associated to each tuple and saves the class label of the tuples. This is composed by two columns: nid and cid. Nid identifies the leaf node meanwhile cid identifies the class label of the tuple. In TALOS they extended this class list with another column, called sid, necessary to support data classification with free tuples. This column provide an additional information necessary to determine the optimal Gini index. We can classify a tuple in three ways:

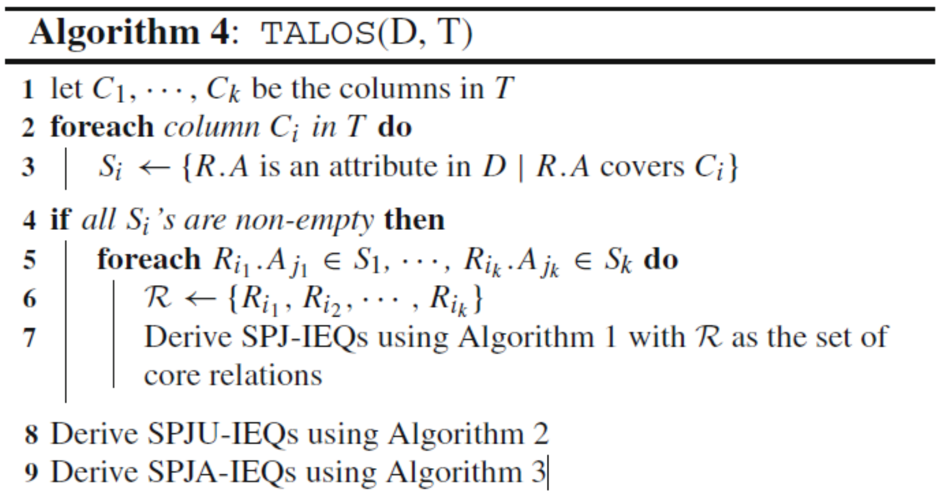
- cid=0 and sid=0: the tuple is a negative one;
- cid=1 and sid=j: the tuple is a bound one;
- cid=-1 and sid=j: the tuple is a free one.

Here j indicates the subset to which the tuple belongs to.

Once they introduced the classification problem and how they solve it with the decision tree classifier, they focused on how to rank the obtained IEQs. They proposed three different methods to evaluates the IEQs, one based on the principle of minimum description lenght and two on the F-measure metrics:

- **Minimum description length (MDL):** the best model is the one that minimizes the sum of the cost of describing the data given the model and the cost of describing the model itself;
- **F-measure:** The first method follows the standard definition of F-measure and a IEQ with an higher value is more precise and a better query. The second one calculates the F-measure not on the actual values but with some data probabilistic models.

Now we can see how they have handled the case when they didn't know the input query, which is the case we have treated in our thesis. They now consider the problem of deriving IEQs given only D and the query result T. In the pseudocode below we can see the approach of TALOS to solve this problem. Before generating SPJ-IEQs, TALOS determines all the possible schema attributes that covers each column in the query result T.



```
Algorithm 4: TALOS(D, T)
1 let $C_1, \cdots, C_k$ be the columns in $T$
2 foreach column $C_i$ in $T$ do
3     $S_i \leftarrow \{R.A$ is an attribute in $D \mid R.A$ covers $C_i\}$
4 if all $S_i$'s are non-empty then
5     foreach $R_{i_1}.A_{j_1} \in S_1, \cdots, R_{i_k}.A_{j_k} \in S_k$ do
6         $\mathcal{R} \leftarrow \{R_{i_1}, R_{i_2}, \cdots, R_{i_k}\}$
7         Derive SPJ-IEQs using Algorithm 1 with $\mathcal{R}$ as the set of
          core relations
8 Derive SPJU-IEQs using Algorithm 2
9 Derive SPJA-IEQs using Algorithm 3
```

Figure 3.3: Pseudocode to find IEQs from an unknown Query

They have also provided an optimized technique to find the covering attributes, called **domain indices**. TALOS creates a three-column mapping table that saves the value, in which schema it is contained and the frequency of the value in that schema. TALOS performs then a join between the result table and this created table and find if an attribute covers a column present in the result table.

Lastly they provided a solution for handling multiple database versions, which we will not explain as it isn't part of our thesis.

This is the main idea behind QRE, how it works and what is the purpose of this algorithm. Our thesis take advantage of these studies by handling the case of the unknown input queries and by obtaining IEQs on which we will apply some functions to generate the desired mapping. We will see in the next chapter, Ch 4, how we will use QRE.

3.2.1 Decision Tree in Qre

Now we can explain how the decision trees are used in the QRE algorithm. We use the decision tree in order to find the attribute on which to split our data set. This algorithm requires a numerical data set and this is why we have used encoding, explained previously in Chapter 2. The attributes that has the best gini index are going to be split on a certain threshold. For example, since in our case we will have many binary columns created by the encoding, it will probably occur that an attribute will be split on a threshold of 0.5 where on the left we will have all the tuples with the value of this attribute equals to 0 and on the right the ones with value equals to 1. If after one split we have a partition that includes all the tuples we are looking for, the algorithm return the attribute and we create a where clause for the query that corresponds to this split. Instead, if the partitions have mixed tuples (some correct and others that aren't), we will recursively call the function and find other attributes to split. Since we want different queries to work with, we cannot split only on one attribute but we have modified the QRE algorithm so that we create **n** (we have put n=3) different splits.

An example: if we are looking for George Clooney as an actor, we could have a split on the attribute "actors*George Clooney" (that is a binary column obtained by the encoding) where the left partition will have all the tuples with value zero in the column "actors*George Clooney" and the right one will have all the tuples with value zero. So in the where clause we will have actors = George Clooney.

Chapter 4

Methodology

In the previous chapters we have summarized all the theory and studies necessary to better understand the problem we have addressed. In this chapter we make a detailed analysis of the methodology followed by the thesis. With the following graph and bullet list we introduce the main steps of our method:

1. Choice of Source and Target dataset;
2. Execution of QRE on the Source dataset obtaining a set of IEQs;
3. Search for all the possible result tables which have a correspondence in the Target;
4. Execution of QRE on the Target with every result table found in the previous step which will give a set or more of IEQs;
5. Evaluation of the Where-clauses from each target set of IEQs comparing them with the ones from the source;
6. Update of the Matrix according to the results;
7. Iteration of these steps until Matrix's values are clear;
8. Creation of the Mapping;

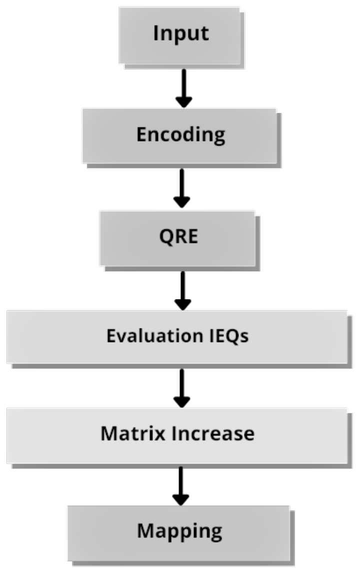


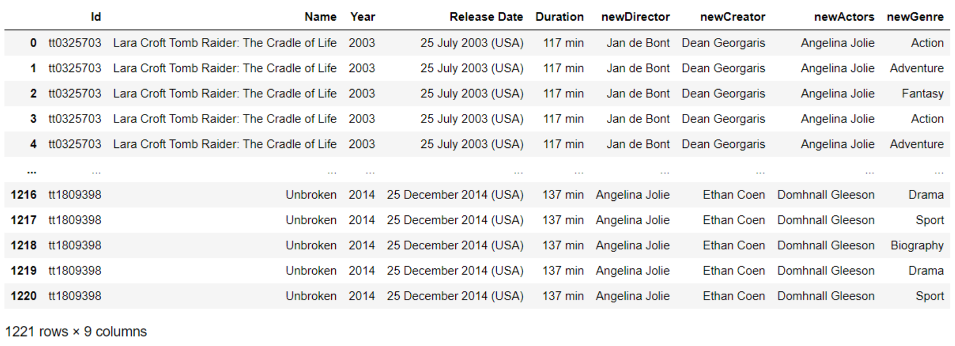
Figure 4.1: Steps of the Methodology

All these steps will be explained in this chapter, focusing on all the reasoning we have made and the choices we have taken.

4.1 Choice of Datasets and Example

Now I will introduce a use case that will be useful to explain all the steps in the methodology. We applied our work on two datasets with movies' domain. The source dataset, coming from rotten tomatoes, composed by eleven attributes (so eleven columns) and 181075 rows. The target dataset, taken from IMDB, composed by eight attributes (so eight columns) and 4346150 rows. (The construction of the datasets is explained in Chapter 5). We have noticed that two attributes ("NewLanguage" and "newCountry") of the source are not present in the target and so we will drop them so we can reduce the number of columns that will created after the encoding. We still have one extra attribute in the source but we will not drop it because we have that "year" and "Release date" in the source are represented by the same attribute "startYear" only that the second one, "Release date", is not only the year but it is the complete date so it should not be mapped to the "startYear" attribute. We want to see if the mapping works correctly even for attributes that are not to be mapped.

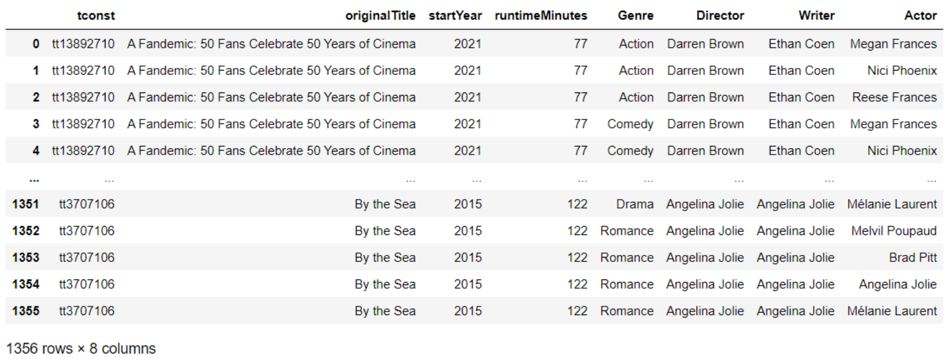
For performance issues we had to take subset of these dataset considering only small parts composed by approximately one thousand or two thousand tuples. For this example we will consider the subsets composed by all tuples where Angelina Jolie appears as actor or director or writer and plus we have added the tuples where Ethan Coen appears as director. Here are the two datasets ready to be read by our function.



	Id	Name	Year	Release Date	Duration	newDirector	newCreator	newActors	newGenre
0	tt0325703	Lara Croft Tomb Raider: The Cradle of Life	2003	25 July 2003 (USA)	117 min	Jan de Bont	Dean Georgaris	Angelina Jolie	Action
1	tt0325703	Lara Croft Tomb Raider: The Cradle of Life	2003	25 July 2003 (USA)	117 min	Jan de Bont	Dean Georgaris	Angelina Jolie	Adventure
2	tt0325703	Lara Croft Tomb Raider: The Cradle of Life	2003	25 July 2003 (USA)	117 min	Jan de Bont	Dean Georgaris	Angelina Jolie	Fantasy
3	tt0325703	Lara Croft Tomb Raider: The Cradle of Life	2003	25 July 2003 (USA)	117 min	Jan de Bont	Dean Georgaris	Angelina Jolie	Action
4	tt0325703	Lara Croft Tomb Raider: The Cradle of Life	2003	25 July 2003 (USA)	117 min	Jan de Bont	Dean Georgaris	Angelina Jolie	Adventure
...	...	...	...	...	...	...	...	...	...
1216	tt1809398	Unbroken	2014	25 December 2014 (USA)	137 min	Angelina Jolie	Ethan Coen	Domhnall Gleeson	Drama
1217	tt1809398	Unbroken	2014	25 December 2014 (USA)	137 min	Angelina Jolie	Ethan Coen	Domhnall Gleeson	Sport
1218	tt1809398	Unbroken	2014	25 December 2014 (USA)	137 min	Angelina Jolie	Ethan Coen	Domhnall Gleeson	Biography
1219	tt1809398	Unbroken	2014	25 December 2014 (USA)	137 min	Angelina Jolie	Ethan Coen	Domhnall Gleeson	Drama
1220	tt1809398	Unbroken	2014	25 December 2014 (USA)	137 min	Angelina Jolie	Ethan Coen	Domhnall Gleeson	Sport

1221 rows × 9 columns

Figure 4.2: Subset from the Source



	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
0	tt13892710	A Fandemic: 50 Fans Celebrate 50 Years of Cinema	2021	77	Action	Darren Brown	Ethan Coen	Megan Frances
1	tt13892710	A Fandemic: 50 Fans Celebrate 50 Years of Cinema	2021	77	Action	Darren Brown	Ethan Coen	Nici Phoenix
2	tt13892710	A Fandemic: 50 Fans Celebrate 50 Years of Cinema	2021	77	Action	Darren Brown	Ethan Coen	Reese Frances
3	tt13892710	A Fandemic: 50 Fans Celebrate 50 Years of Cinema	2021	77	Comedy	Darren Brown	Ethan Coen	Megan Frances
4	tt13892710	A Fandemic: 50 Fans Celebrate 50 Years of Cinema	2021	77	Comedy	Darren Brown	Ethan Coen	Nici Phoenix
...	...	...	...	...	...	...	...	...
1351	tt3707106	By the Sea	2015	122	Drama	Angelina Jolie	Angelina Jolie	Mélanie Laurent
1352	tt3707106	By the Sea	2015	122	Romance	Angelina Jolie	Angelina Jolie	Melvil Poupaud
1353	tt3707106	By the Sea	2015	122	Romance	Angelina Jolie	Angelina Jolie	Brad Pitt
1354	tt3707106	By the Sea	2015	122	Romance	Angelina Jolie	Angelina Jolie	Angelina Jolie
1355	tt3707106	By the Sea	2015	122	Romance	Angelina Jolie	Angelina Jolie	Mélanie Laurent

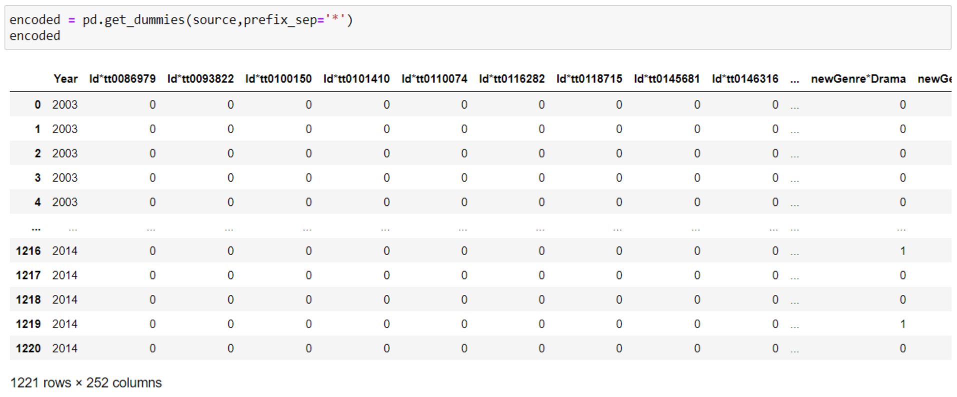
1356 rows × 8 columns

Figure 4.3: Subset from the Target

As we can see from the figures above, we can spot the attributes that are representing the same values but an algorithm cannot spot this relationships. So we want to apply our method to obtain all the possible mappings between the attributes in the two subsets. For example, at the end of our procedure we will expect that the attribute "Name" in source will be linked with the attribute "originalTitle" in the Target. We can also notice how small these datasets are in confront with the original ones and that it isn't necessary for the subsets to have all the same tuples.

Prior to the execution of the QRE algorithm on the datasets we must do some preprocessing step. We have described some steps in the previous chapter when we have described how we have obtained the datasets. In this chapter we will focus more on the lasts steps and how the data must be structured. Our paper uses the QRE algorithm from [1] which is based on the paper Query by Output (Tran et al.) described in the state of art chapter. This algorithm does not support datasets with categorical data. Since our studies will have categorical data, we had to implement an encoding function, as explained in Chapter 2, to transform our categorical datasets in numerical datasets. Here we can see how our datasets change after the encoding has been applied to them, I purposely left the dimensions of the datasets in the picture so we can see how the number of columns has increased after applying panda.get_dummies(). We have modified the base prefix "_" with "⋆" because some titles had _ in their names and we need a specific character to be able to separate the new string and return to our original values when we will create the where-clauses.

[1]https://github.com/drblallo/QueryByOutput

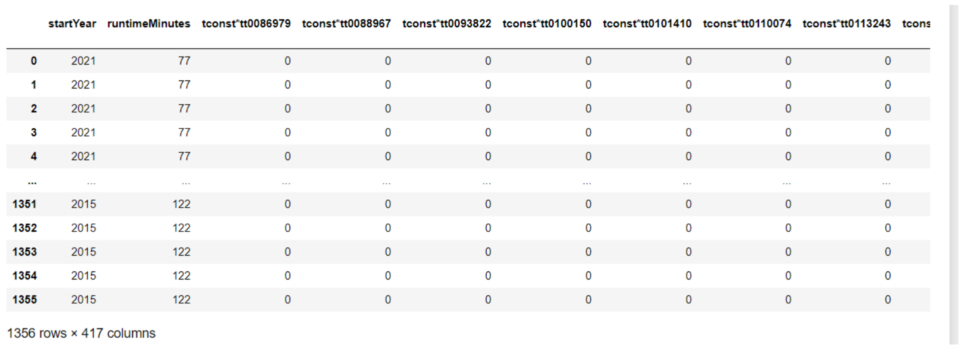


```
encoded = pd.get_dummies(source,prefix_sep='*')
encoded
```

	Year	Id*tt0086979	Id*tt0093822	Id*tt0100150	Id*tt0101410	Id*tt0110074	Id*tt0116282	Id*tt0118715	Id*tt0145681	Id*tt0146316	...	newGenre*Drama	newGe
0	2003	0	0	0	0	0	0	0	0	0	...	0	
1	2003	0	0	0	0	0	0	0	0	0	...	0	
2	2003	0	0	0	0	0	0	0	0	0	...	0	
3	2003	0	0	0	0	0	0	0	0	0	...	0	
4	2003	0	0	0	0	0	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1216	2014	0	0	0	0	0	0	0	0	0	...	1	
1217	2014	0	0	0	0	0	0	0	0	0	...	0	
1218	2014	0	0	0	0	0	0	0	0	0	...	0	
1219	2014	0	0	0	0	0	0	0	0	0	...	1	
1220	2014	0	0	0	0	0	0	0	0	0	...	0	

1221 rows × 252 columns

Figure 4.4: Source Encoded



	startYear	runtimeMinutes	tconst*tt0086979	tconst*tt0088967	tconst*tt0093822	tconst*tt0100150	tconst*tt0101410	tconst*tt0110074	tconst*tt0113243	tcons
0	2021	77	0	0	0	0	0	0	0	
1	2021	77	0	0	0	0	0	0	0	
2	2021	77	0	0	0	0	0	0	0	
3	2021	77	0	0	0	0	0	0	0	
4	2021	77	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
1351	2015	122	0	0	0	0	0	0	0	
1352	2015	122	0	0	0	0	0	0	0	
1353	2015	122	0	0	0	0	0	0	0	
1354	2015	122	0	0	0	0	0	0	0	
1355	2015	122	0	0	0	0	0	0	0	

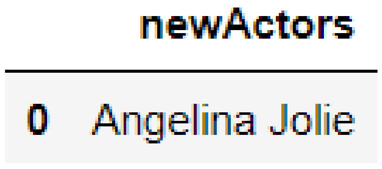
1356 rows × 417 columns

Figure 4.5: Target Encoded

Once we have encoded our datasets we can apply the QRE algorithm seen in Chapter 3. We will first apply it on the Source dataset of which we have the complete **result** with both the value of the attribute and the name of the attribute (name of the column). Then we will perform the QRE algorithm on the Target dataset but in this case we will not know the name of the attribute but only the value.

Here we have nominated the **result** which is the output obtained from an unknown query that we have has input in our QRE and from which we want to derive the IEQs that could have created it. In the source case this is a two rows dataset where the first row has the name of the attribute (or attributes) that are returned and the second row the searched value (or values). In the target case we will take only the second row, the values of the attributes. From these values we will obtain the combination of all attributes in the target that could represent these value and then we will create our complete result with one row representing the attributes and the other one with the values.

In our example, we will look for the tuples were Angelina Jolie is the actor so our result in input for the source will be like:



	newActors
0	Angelina Jolie

Figure 4.6: Example on the source

It is evident that this example is on a categorical attribute and so, as we said before, since our QRE algorithm wants only numerical attribute we have to encode also this example before we can perform the research. The target example will be created looking at the values of the attributes, in this case "Angelina Jolie". We will see later how we have obtained all the possible examples in the target from one value.

4.2 Applying QRE on Source

Now that we have the subset 4.4 and the example 4.6 we can apply the QRE algorithm. The first step will create two lists from the source subset, one with all the name of the columns and one with all the values present in the subset, the same step is done also for the example. So we will have four lists, two with the names of the columns and two with all the values.

The next step will create a new table from which the algorithm will generate the IEQs. This table is obtained by confronting the columns and rows of the subset and the example. In this table it is added a column that indicates the class of each row, the class is specified by values:

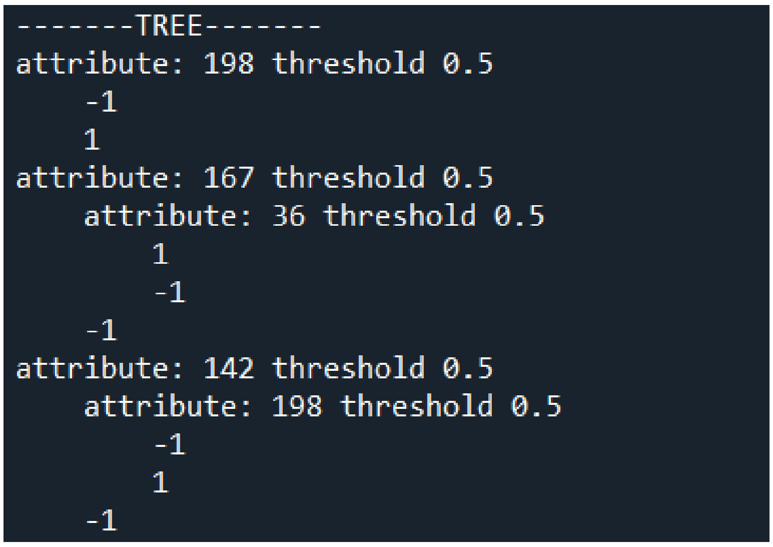
- -1: the row doesn't project any row of the example (bound negative);
- 0: if the row project a row in the example but it is not the only one (free);
- 1: if the row project a row in the example and it is the only one (bound positive);

Now we have a numerical table with all the row classified positive, negative or free. We can call our decision tree to find the optimal splits. In our algorithm we have adoperated the first case presented in the QRE algorithm in Chapter 3, the one where all the free tuples are considered as positive.

Initially the algorithm was built with the goal to find one optimal split looking at the Gini index and splitting on the minimum and so it would have returned a single tree that would have generated a single IEQ. For our purpose we wanted to have more than one IEQ returned and so we have modified the algorithm in a way that it will create three tree. Our algorithm calculates the Gini index of every attribute in the table and return all the Gini with all the possible split, then we select only the three best Gini and iterate only on those splits creating indeed three tree. To do so we have modified the function that returns the optimal split in such a way that returned all the possible splits as generators. Then we iterate on these generator and take only the three best one.

In the main function once we have called the decision_tree function we will see as result a generator on which once we iterate we obtain a list of three (in our case we will have three tree). These tree will have as information the attribute (saved as the number of column) on which they have done the split, the threshold on which the attribute is been split and a number, -1 or 1, that indicates if the taken split will be relevant as a solution or if it doesn't have positive tuples in it.

Here an example of the trees generated from our use case on the source.



```
-------TREE-------
attribute: 198 threshold 0.5
    -1
    1
attribute: 167 threshold 0.5
    attribute: 36 threshold 0.5
        1
        -1
    -1
attribute: 142 threshold 0.5
    attribute: 198 threshold 0.5
        -1
        1
    -1
```

Figure 4.7: Example of the trees created from the source subset

As we can see we have three different trees that split on different attributes. The first one is a simple three that splits on one attribute and the left three doesn't have positive tuples (-1 as value), instead the right tree has positive tuples (1 as value). The other two are more complex, they have an initial split on an attribute but then they have another split on another attribute on the left tree. We can see that the only feasible split (the only split with 1 as value) is inside the second split so we have two condition to be verified, the first and the second split, we have an "AND" condition. If instead we had also a 1 value in another split we would have obtained an "OR" condition.

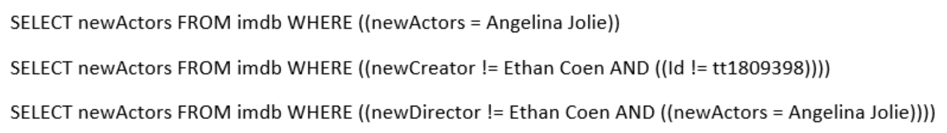
Now we have the trees and we can call for every tree the function that create the IEQs from a tree. This function, **tree_to_query()**, take as input the names of the attributes (the ones in the example table), the name of the tables from which we are taking the tuples, the columns of the subset and a tree. The query is formed like that:

- **SELECT:** the select-clause is created from the names of the attributes, it simply return a string with "SELECT" and the names of the attributes selected;
- **FROM:** the from-clause simply return a string with "FROM" and the name of the tables from which the attributes are taken;
- **WHERE:** the where-clause recursively builds the logical formula given the decision tree;

The first two clauses are quite easy and intuitive, we now focus on the third one, explaining also how the logical operators are added in the predicates of the where-clause. First of all it checks that the tree is not a leaf node, if it is it checks if it is a positive one (if tree=1) or a a negative one. If it is positive return True, else it return False. After this check, the algorithm start to build the query. It is obvious that once we enter this function the tree has at least one split and so a left tree and a right tree. The function call recursively itself on the left tree and on the right one until the first check return true or false, we have reached a leaf node. So we are building the condition starting from the inner predicates. Once we have reached a leaf node, if it is true we build the predicate. If "*" is in the attribute name on which we have the split, we have an encoded attribute (a categorical one) and we have to reverse encode it to obtain the correct attribute name and value. If we are in the left tree we reverse encode it with the condition "!=" instead in the right one we have the "=" condition. On the left part of the operator we put the name of the attribute, on the right one the value of the attribute. Those are obtained by dividing the given attribute name, the substring before "*" is the original attribute name, the one after is the value of the attribute. If we don't have "*" in the attribute name we have a numerical attribute which was not encoded so we have that the original attribute is the one on which we have split and the threshold is the value of the attribute on which we have split. Here the condition will be on the left tree the attribute must be ”<=” of the value of the threshold and in the right one it must be ”>” of the threshold.

This is how we obtain a predicate, but in a where clause we can have more then one predicates that are bounded by the logical operators ”AND” and ”OR”. In our algorithm we have an ”AND” when a subtree is not a leaf node because in this subtree we will have a split with positive tuples and so we will need at least two condition verified (the first split and the second one) which are defined with an ”AND” operator. Instead, we have an ”OR” when both the left subtree and right subtree have a positive split so we need to consider one condition or the other one which is the definition of the ”OR” operator.Once the function end the recursion on itself we will have the where-clause with the condition composed by one or more predicates.

We can see an example of IEQs obtained by the trees showed in 4.7, in the picture below:



```
SELECT newActors FROM imdb WHERE ((newActors = Angelina Jolie))
SELECT newActors FROM imdb WHERE ((newCreator != Ethan Coen AND ((Id != tt1809398))))
SELECT newActors FROM imdb WHERE ((newDirector != Ethan Coen AND ((newActors = Angelina Jolie))))
```

Figure 4.8: Example of the queries created from the source trees

Keeping in mind the generated tree we explain the IEQs obtained from them. In this case we don’t have numerical attribute but only categorical ones, we see for example that the first query is a condition with one predicate since the tree has only one split as we noted previously. The positive split was the right one so as condition we need to have ”=” then we have that the attribute 198 must have been ”newActors*Angelina Jolie” so we have that the original attribute was ”newActors” and the value was ”Angelina Jolie” so here explained the predicate in the first query. Now we will explain the second query just to see the case of ”AND” operator. We have an initial split on attribute 167 that gives a right tree with only negative tuples, and a left tree with positive tuples after another split on attribute 36. So we will have that one condition will be the one given by the split on attribute 167, ”newCreator*Ethan Coen” and the other one the one given by the split on attribute 36 ”Id*tt1809398”. We have two condition on the same branch so we have the ”AND” operator. Since the first split is on the left tree and the second split has the positive tuples on the left tree we will have has predicate two ”!=”. The final condition will be the ”AND” operator between two predicates with the ”!=” condition, as we can see in 4.8.

Now we have obtained the IEQs from the source and we can move to the target.

4.3 Search of possible example in the Target

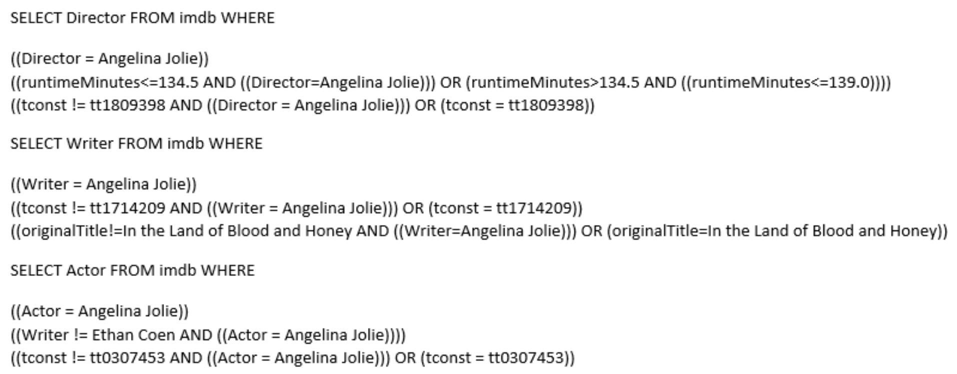
With the source we have applied QRE knowing what attribute we were looking for, now with the target we dont have this information. In fact, our goal is to find all the possible correspondences in the target of the example previously used for the source. So, before we can give the target dataset and the example as input to the QRE we have to build the example. We perform a complete scan of the target searching in every column the value we are looking for. We save in a list the lists of attribute names that have an occurrence of the value. Then we have to combine those lists with a cartesian product to create all possible examples.

In our use case we have that the value we are looking for is ”Angelina Jolie” and in our target subset it appears in three columns:

- **Director:** when Angelina Jolie appears in a tuple as director of a film;
- **Writer:** when Angelina Jolie appears as a writer of a film;
- **Actor:** when Angelina Jolie appears as an actor of a film.

We know that we are looking for the tuples where Angelina Jolie is the actress but our method cannot know a priori what is thee correct column to take, so once we have all the possible columns we create different examples and for each one of them we call the process function.

Now we are in the same situation as for the source, we have a subset (the target) and an example with an attribute name and a value. We must encode these datasets, so we apply pd.get_dummies, and then we give the encoded target and the example as input to the QRE algorithm. The QRE works as before, only this time will be called one time for every example we have. At the end we will obtain three set of trees from which we will extract respectively three set of IEQs. The IEQs obtain in our use case are:



```
SELECT Director FROM imdb WHERE

((Director = Angelina Jolie))
((runtimeMinutes<=134.5 AND ((Director=Angelina Jolie))) OR (runtimeMinutes>134.5 AND ((runtimeMinutes<=139.0))))
((tconst != tt1809398 AND ((Director = Angelina Jolie))) OR (tconst = tt1809398))

SELECT Writer FROM imdb WHERE

((Writer = Angelina Jolie))
((tconst != tt1714209 AND ((Writer = Angelina Jolie))) OR (tconst = tt1714209))
((originalTitle!=In the Land of Blood and Honey AND ((Writer=Angelina Jolie))) OR (originalTitle=In the Land of Blood and Honey))

SELECT Actor FROM imdb WHERE

((Actor = Angelina Jolie))
((Writer != Ethan Coen AND ((Actor = Angelina Jolie))))
((tconst != tt0307453 AND ((Actor = Angelina Jolie))) OR (tconst = tt0307453))
```

Figure 4.9: IEQs obtained from the target

We can notice how for every attribute we found before we have a different set of IEQs.

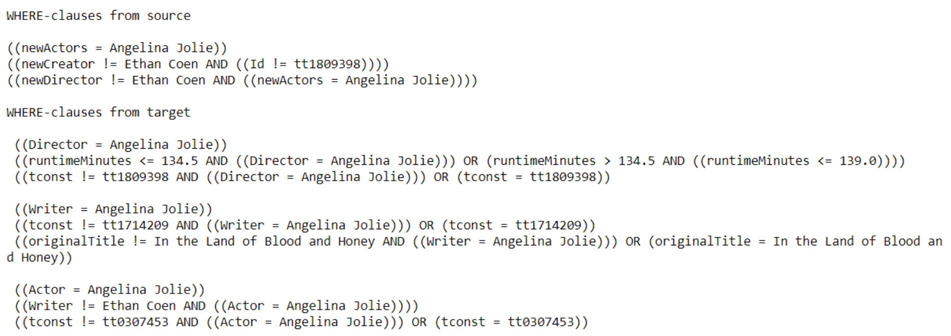
4.4 Analisys of Where-clauses

Now that we have the lists of queries for every attribute, we need to compare the where clauses with the queries of the source and find the set of IEQs that are more similar. Specifically, when we will find the set of queries that look like the queries from the source, we can hypothesize that the attribute in the select clause of the target and of the source is representing the same object. We have to save this result but we cannot conclude just with one example that this is correct, we need more examples and a sum of all the results to be able to have a more correct matching.

To do so we have built a NxM matrix where each column is a target attribute and each row is a source attribute. Reasoning behind this matrix explained in the next section.

Our goal is to increment the values in the matrix but before that, as we have previously said, we have to find the set of queries (one set from the source and one set from the target) on which perform these evaluations. The function that we have created will take as input one where-clause from the source and a list of where-clauses from the target. At this moment we have a list with three queries from the source and a list of queries lists from the target. We need to take these queries and split them on the where-clause keeping only the part where we have the actual condition. To do so we use the split() function that applied on a string and given a word on which to split, separates the string and return a list with the left string and right string. We save only the right one. For the source is quite simple, we run a for on the length of the list and for each query we use the split function. For the target we need a double loop, one to loop on the lists inside the list and the other one to loop on the queries inside each list. At the end we will have the same lists but with only the conditions inside the where clauses instead of the complete queries. Now we can explain our function and how we have built the comparison between the where clause.

In our use case after these splits our list of queries will be like:



```
WHERE-clauses from source

((newActors = Angelina Jolie))
((newCreator != Ethan Coen AND ((Id != tt1809398))))
((newDirector != Ethan Coen AND ((newActors = Angelina Jolie))))

WHERE-clauses from target

 ((Director = Angelina Jolie))
 ((runtimeMinutes <= 134.5 AND ((Director = Angelina Jolie))) OR (runtimeMinutes > 134.5 AND ((runtimeMinutes <= 139.0))))
 ((tconst != tt1809398 AND ((Director = Angelina Jolie))) OR (tconst = tt1809398))

 ((Writer = Angelina Jolie))
 ((tconst != tt1714209 AND ((Writer = Angelina Jolie))) OR (tconst = tt1714209))
 ((originalTitle != In the Land of Blood and Honey AND ((Writer = Angelina Jolie))) OR (originalTitle = In the Land of Blood an
d Honey))

 ((Actor = Angelina Jolie))
 ((Writer != Ethan Coen AND ((Actor = Angelina Jolie))))
 ((tconst != tt0307453 AND ((Actor = Angelina Jolie))) OR (tconst = tt0307453))
```

Figure 4.10: Where-clauses after the splits

As we said the function takes as input a condition of a source query and one of the lists in the list of condition of the target. So, we have created a double loop. An external for-loop that loops on the list of the target and an internal for-loop that loops on the condition of the source. When we have a condition (the where-clause) from the source and a list of conditions from the target we have to look at the condition from the source before we can call the function.

4.4.1 Logical Operators

The where-clause could present different predicates bounded with logical operators as or and and, or it could also be without logical operators. If we have a logical operator, we need to separate the predicates bounded by the operator and at the end perform a **summative calculation**.

These are the possibilities:

- We have the OR operator; We need to split the condition on the OR obtaining two separated strings that taken alone will represent a predicate. Then we have to look at those strings, we could encounter two possible cases. We have an AND or we dont have logical operators. In the latter one we simply call our function that will give us the result of the comparison that we will save in a list (resFirst). In the first one we will have to split on the AND operator and so, we will have other two strings and for both we need to call the function. We save the result of the comparison of both strings in a list (partialFirst) and then we append the average in another list (resFirst). So at the end we will have 2 result in the resFirst list, one for the constraint on the right and one for the constraint on the left (the predicates obtained by splitting on the OR). Since we are dealing with an OR that return true if only one of the who condition is verified, we have decided to consider as result the max value present in the resFirst. So, we have appended in another list (totFirst) the max value present in resFirst.

- We have the "AND" operator; here we must split on the "AND" and we will obtain two separated predicates which will be without logical operators and ready to be passed to the function one by one. Since the "AND" operator is true only when both conditions are verified, we have chosen to take the results given by the function saved in a list(partialFirst), make the average and save it in another list (totFirst).
- We don't have logical operators; in this case we don't have to split the string before passing to the function. When we call the function we will obtain directly the result we want and save it in (totFirst)

This was the description of all the possible cases of how the conditions in the where clause of the source query could appear. Now we have to look at how every predicate is handled by the function, since we could have different cases.

For example, referring to 4.10, we can see that in the source we have the first where-clause which is already written as a simple predicate and we can call the function directly passing as input this predicate. Instead, in the other two where-clauses we have and "AND" operator so we need to separate the condition on the "AND" and we will obtain two different predicates for each where-clause on which we have to call the function.

4.4.2 Studying the Predicates

Once we called the function, we can start to look at the mathematical symbols that appears in the predicate of the source string. Here we have to explain in details every case because depending on each symbol we have to take different path in the code. One thing should be clear by now, we have called the function with the goal to find the max value of comparison between the source predicate and the three conditions present in the list of where-clauses of the target. So, in the target we could still have conditions with one or more predicates and we will have to split them as we have done with the source. Then we can confront the source predicate with a single predicate and so on. I will not discuss this in detail since it is pretty similar with the reasoning done for the source. There is only one thing that is different and it is how we handle the result of an "AND" operator. In the source we have decided to take the average of the results obtained by the two predicates bounded with the "AND" operator because we thought that both predicates should have counted in the final result. Instead, now we are comparing a predicate from the source with a condition of the target. This condition could have an "AND" and so we will split it into two predicates. So, now we have one predicate from the source that compares with two predicates from one condition of the target. It is logical to think that the source one could not be equal or similar at the same time to the two target predicates. So, we have decided to take as result only the max result we get from the two comparison and not the average.

We have made an assumption when we have a categorical attribute, we can only compare predicates with the same operator. For instance, if we have "!=" in the source one we need to have != in the target predicate.

We have four possibilities:

- We have the **!=** symbol. We split the first string on the mathematical symbol and then we create the loop described before. When we have the predicates of the target, we check that they have the "!=" symbol and in that case we call the fuzz.ratio() function passing as input the source predicate and the target predicate.

- We have the = symbol. We create an if to control that "<" is not in the predicate and then split on the equal operator. Create the predicates from the target list and then before we can call the fuzz ratio we check that the target predicate has the "=" symbol and that it does not have "!=" or "<". Call the fuzz ratio function and return the result in the list.

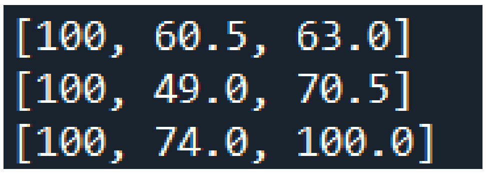
- We have the <= symbol. This is the else case from the previous if, we have an equal but also the < symbol. In this case we are dealing with a numerical attribute in the source string and we are going to look for a numerical attribute also in the target string or we will discard the comparison. As before we split on the "=" symbol and we also eliminate the black space so we will have only the integer in the string. Then we obtain the predicates from the target as before and once we got the predicates we check that they are a numerical attribute looking for a $<$ or $>$ symbol. If we don't have a numerical attribute we return as value of the comparison a zero, else, we will have different interval and we will have to make some evaluations. Since in the source we had $<$, we have assumed that if also in the target we have $<$ we can return 100 as result of the comparison since one interval contains the other. If in the target we have $>$, we need to look at the values of the integers. If the source value is lower then the target value we do not have an intersection of the interval so we return 0, else, we have an interval and we return 100 (further studies on what to return should be made)

- We have the $>$ symbol. We have a numerical attribute in the source target and as before we will compare it only with another numerical attribute taken from the target. The only difference from the previous case is how we will have the intervals between the values. If we have $>$ also in the target, as before one interval contains the other so we return 100. Else, we will have that if the source value is lower then the target one we have an intersection and we return 100, or we will return zero when there is no intersection.

4.4.3 Evaluation of Results

Once we have called the function and studied all the possible comparison between the queries we have as result a list of lists composed by some values. One list with these values is representing the result of the comparison between the source set of queries and a set of queries from the target. So, we need to take the list with the max values between all the lists we have obtained. This list will correspond to the set of queries from the target that is more related to the ones of the source and we could conclude that the attribute in the select clause between the two set of queries is matching. To find the list with the higher values we initially sorted all the value in the lists. Then with a while we will look at which lists has the higher values. After this while we will know which set of queries is the one to take to generate the mapping and we can know increment our matrix.

Here we can see some of these steps applied to our use case. We were at the point where we have obtained the where-clauses of all the IEQs, 4.10. We are expecting that the IEQs from the source which have the attribute "newActors" in the SELECT-clause will be more similar to the IEQs from the target that have "Actor" as SELECT-clause. The first step will be to compare the IEQs of the source with the first set of IEQs of the target, the ones that select on the "Director". Then it will compare with the ones with "Writer" as SELECT attribute and lastly with the ones that have "Actor" as SELECT attribute. In our use case the result of these comparison will be equal to:



```
[100, 60.5, 63.0]
[100, 49.0, 70.5]
[100, 74.0, 100.0]
```

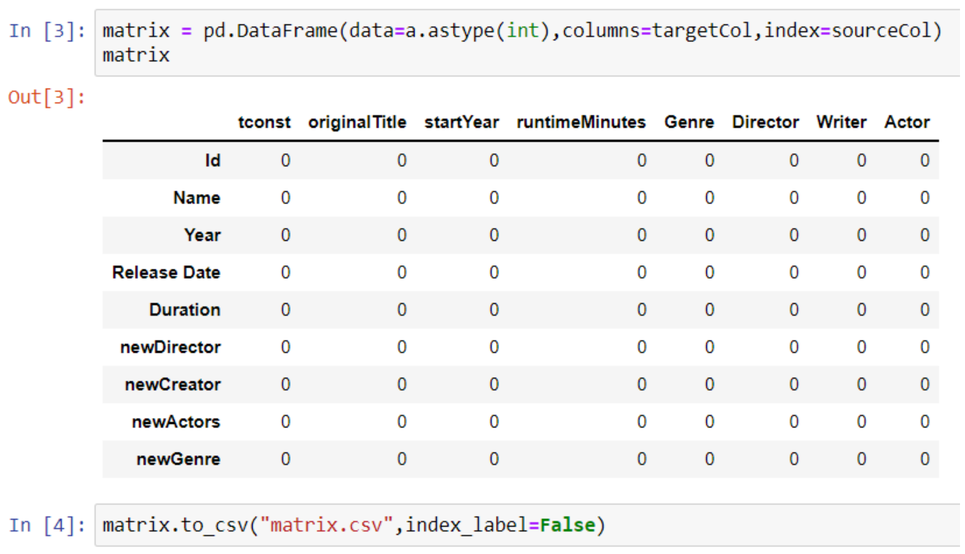
Figure 4.11: Results from the comparison of the where-clauses

We can see that the third one has two where-clauses that compare with a result of 100%, instead in the other ones we have only one where-clause that does so. This was what we were expecting since both set of IEQs, the one from the source and the one from the target, want to select an actor. Before we introduce the matrix, we take a look again at 4.10 and with this previous step we can also understand which where-clauses generated those results. We have that the first where-clause of the source with the first one of the target search for an attribute with the value "Angelina Jolie", so they both looked for the same value. Then we have that the third where-clause of the source is returned a 100% value from the result. We can see that it is given with the comparison with the second where-clause of the target. In both clauses we have an "AND" operator so we need to split the conditions in two different predicates. Now we have one predicate from the source, for example "newCreator != Ethan Coen", that will compare with the two predicates of the target and will return the best comparison. We can notice that one of these predicates has the "!=" operator and search for the same value of the source so here we have the 100% comparison result. We can notice how the attribute on which "!=" is applied is not the same in the source and in the target, but in this step we are not mapping the attributes so we are not concluding right away that "newDirector" is mapped with "Writer" which would be wrong. This is the reason why we have built the matrix, we cannot base our mapping on only one example because we can obtain wrong results. Now we will see how we have created the matrix and how it works.

4.5 Creation of the Mapping

In the previous section we have introduced the concept of the matrix without going in further details. Here we will have a closer look and explain our final results. What is the reasoning behind this matrix?

We have two datasets with N and M attributes and we want to find the correspondences between these attributes to create our mapping. So, we have built a NxM matrix with the target attributes on the columns and the source attributes on the rows. This matrix is saved in a csv file and it is built external to the code. It is initialized to all values equals to zero. This is how the matrix in our use case would look like:



```
In [3]: matrix = pd.DataFrame(data=a.astype(int),columns=targetCol,index=sourceCol)
        matrix
```

Out[3]:

	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
Id	0	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0	0
Year	0	0	0	0	0	0	0	0
Release Date	0	0	0	0	0	0	0	0
Duration	0	0	0	0	0	0	0	0
newDirector	0	0	0	0	0	0	0	0
newCreator	0	0	0	0	0	0	0	0
newActors	0	0	0	0	0	0	0	0
newGenre	0	0	0	0	0	0	0	0

```
In [4]: matrix.to_csv("matrix.csv",index_label=False)
```

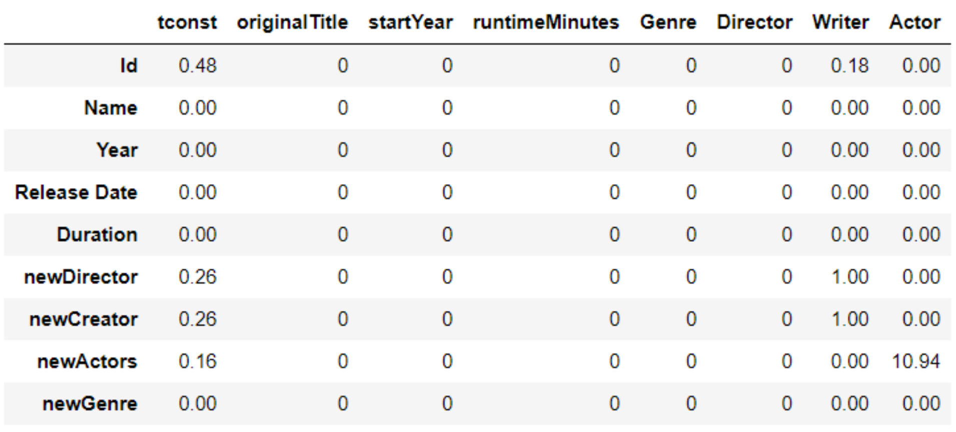
Figure 4.12: The initialized Matrix

We want to increment these values when we find a matching between the attributes. When we execute our code, at the end we will obtain an incremented matrix where some values has been incremented according at the result of the comparison of the where clauses. Since the matrix is external to the code, when we execute our code ten times, we will obtain a matrix incremented ten times without losing our previous results. So, at the end we can run a loop on the columns (or on the rows) and looking at the higher value in each column we can return the mapping between the two datasets.

4.5.1 Incrementing the Matrix

How do we increment this matrix? At this point we have decided which set of queries from the target need to be compared with the one of the source so we will discard the others and consider only this one. We can initially suppose that the attribute in the select clauses should represent the same attribute in both datasets so we can do an increment on the value of the matrix corresponding to those two attributes. We select the value where the row corresponds to the source attribute and where the column corresponds to the target attribute and we increment it by a certain amount stating that those two attributes should be equivalent. We did not stop only on the attributes in the select clauses but we have made a study also on the attributes in the where clauses of those queries. As we did before, we need to confront not the whole conditions in the where clauses but only the predicates in them. So, we built our code similar at when we have analyzed the where clauses in order to compare only the predicates. What we have done different is that when we obtain our result of the comparison we do not save it in a list to be returned, but we call another function, increaseMatrix, that will increment the matrix by a given amount. When we split the predicates we have also saved in another string the value of the attribute and not only the value of the attribute. This function requires as input the name of the attribute in the source predicate, the name of the attribute in the target predicate and the result of the comparison between the values of those attributes. Then with one for loop on the columns and one on the rows of the matrix we search the corresponding column and row for those attributes. We then increment the value in this specific column and row by the percentage of the result obtained by the previous comparison. Update the csv file and we have our incremented matrix.

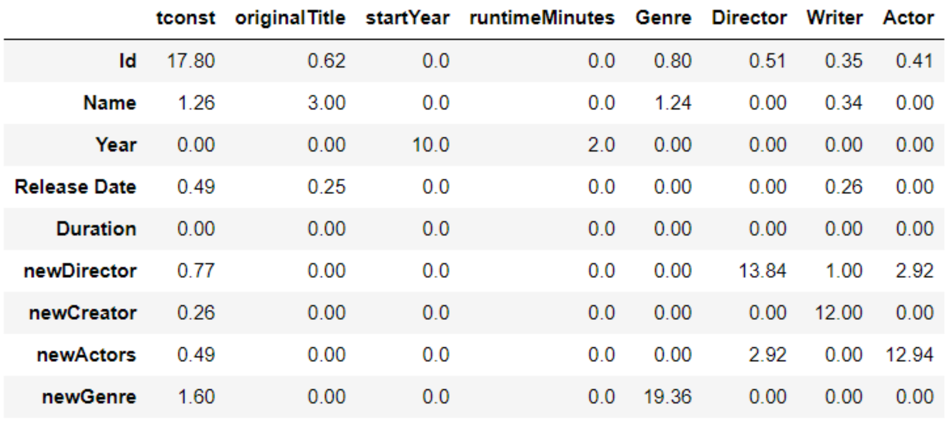
In our use case, after the results obtain in the previous sections, we can build our matrix and obtain this incremented matrix:



	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
Id	0.48	0	0	0	0	0	0.18	0.00
Name	0.00	0	0	0	0	0	0.00	0.00
Year	0.00	0	0	0	0	0	0.00	0.00
Release Date	0.00	0	0	0	0	0	0.00	0.00
Duration	0.00	0	0	0	0	0	0.00	0.00
newDirector	0.26	0	0	0	0	0	1.00	0.00
newCreator	0.26	0	0	0	0	0	1.00	0.00
newActors	0.16	0	0	0	0	0	0.00	10.94
newGenre	0.00	0	0	0	0	0	0.00	0.00

Figure 4.13: Example of the increased Matrix on our use case

In this picture, 4.13, we can notice that some attributes have received some increments, in particular the value obtained by the intersection of attributes "newActors" and "Actor" is sensing that those two attributes represents the same attribute which is true in our case. We can also notice that not all attributes are precisely mapped and in particular we have that the attribute "Writer" from the target has the same value both for "newDirector" and "newCreator". This is the reason why we need more then one single example to create a correct mapping. Here we can see an incremented matrix obtained after the run of five different examples.



	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
Id	17.80	0.62	0.0	0.0	0.80	0.51	0.35	0.41
Name	1.26	3.00	0.0	0.0	1.24	0.00	0.34	0.00
Year	0.00	0.00	10.0	2.0	0.00	0.00	0.00	0.00
Release Date	0.49	0.25	0.0	0.0	0.00	0.00	0.26	0.00
Duration	0.00	0.00	0.0	0.0	0.00	0.00	0.00	0.00
newDirector	0.77	0.00	0.0	0.0	0.00	13.84	1.00	2.92
newCreator	0.26	0.00	0.0	0.0	0.00	0.00	12.00	0.00
newActors	0.49	0.00	0.0	0.0	0.00	2.92	0.00	12.94
newGenre	1.60	0.00	0.0	0.0	19.36	0.00	0.00	0.00

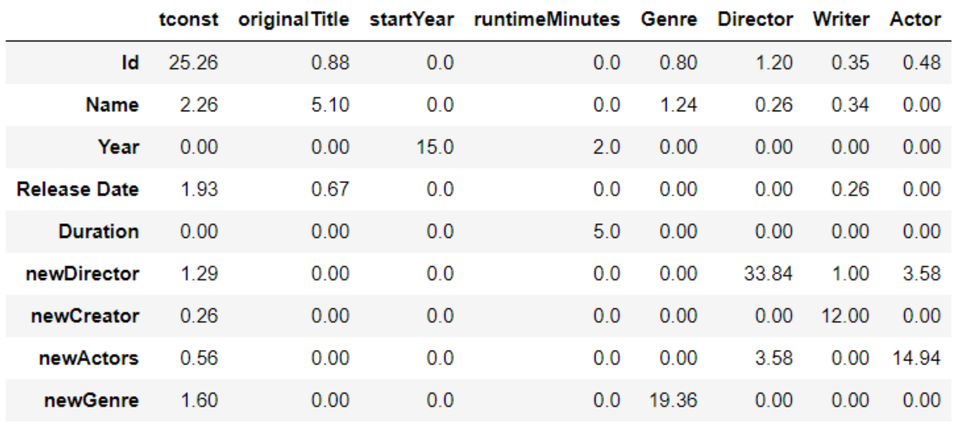
Figure 4.14: Example of increased Matrix after 5 execution

In this last matrix we have a more defined mapping where most of the attributes are mapped correctly and we don't have doubtful cases. From this matrix we could extract a partially correct mapping, we are still missing some attributes for which we will need to run more examples.

4.5.2 Mapping from the Matrix

How do we extract the correct mappings from the matrix? We need to create a function that reads all the rows and columns of the matrix and understand which is the maximum value possible for an attribute both from the source and the target. Once we have created this function we can evaluate our mapping. Before we can do that we created a numpy matrix from the csv file containing our incremented matrix. Once we have this numpy matrix we start with a for-loop on the rows then we have another for-loop on the columns where we checks for each row which is the maximum value. Once we have scanned all the values in the row we check that the found maximum is also the maximum value in the respective column, if it is we link the index of the row with the index of the column saving in a list the index of the column at the respective position of the current row. We repeat this process until the first for-loop comes to an end. Once we have ended our for-loop we can evaluate our mapping. We have the created list with all the linked attributes and we can compare it with a pre-made list were we have saved the expected results.

Here we can see the results obtained by calculating the mapping on a matrix generated after different runs of the method on different examples:

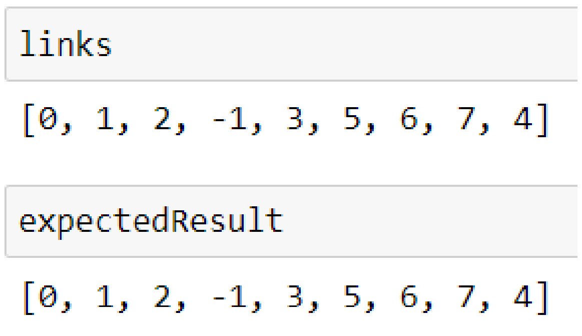


	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
Id	25.26	0.88	0.0	0.0	0.80	1.20	0.35	0.48
Name	2.26	5.10	0.0	0.0	1.24	0.26	0.34	0.00
Year	0.00	0.00	15.0	2.0	0.00	0.00	0.00	0.00
Release Date	1.93	0.67	0.0	0.0	0.00	0.00	0.26	0.00
Duration	0.00	0.00	0.0	5.0	0.00	0.00	0.00	0.00
newDirector	1.29	0.00	0.0	0.0	0.00	33.84	1.00	3.58
newCreator	0.26	0.00	0.0	0.0	0.00	0.00	12.00	0.00
newActors	0.56	0.00	0.0	0.0	0.00	3.58	0.00	14.94
newGenre	1.60	0.00	0.0	0.0	19.36	0.00	0.00	0.00

Figure 4.15: Final increased Matrix

We can clearly see from the picture the generated mappings, how the target attributes are linked with the ones from the source. In this matrix we can see for example that the attribute ”Id” of the target must be linked with the attribute ”tconst” from the source which is the desired link since both attribute represent the code of the film.

As said before I've implemented an automatic way to get all the links and we can notice in the picture below that the obtained results reflect the expected results:



```
links
[0, 1, 2, -1, 3, 5, 6, 7, 4]
expectedResult
[0, 1, 2, -1, 3, 5, 6, 7, 4]
```

Figure 4.16: Result of the mapping

We can see that the two arrays are equal so we have obtained a correct mapping. The value -1 is referring to the attribute "Release Date" which, as we said before in the first section, should not be mapped to any source attribute and also as we see in 4.15 it doesn't have values that provide a mapping with any attribute. So, lastly, our method provided a correct mapping that needs different run on different example in order to be trustworthy and complete.

Chapter 5

Datasets and Experiments

5.1 Creation of Datasets

In this chapter we will explain where we have taken our datasets and how we have modified them to be suitable for our thesis. This thesis work needed two datasets that had the same domain so that they had tuples in common in order to be able to create a mapping between the attributes. For an external user the mapping between the attributes could have been easily done, but for an unsupervised algorithm the mapping is trivial not knowing a priori the meaning of the names of the attributes. We have searched for two datasets with those prerequisites, we have found the rotten tomatoes dataset and the imdb dataset. In the next pages we explain how we have obtained and modelled them.

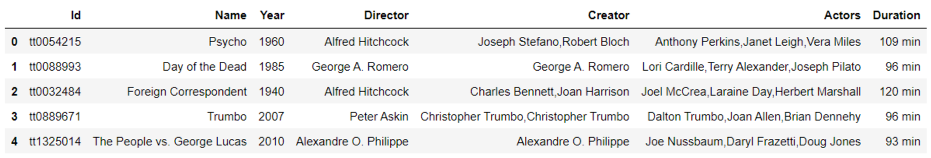
5.1.1 Source Dataset

I have found the source dataset in this repository [1], the rotten_tomatoes.csv file. Before we could use this dataset, we had to make some adjustments. First we have eliminated some attributes (some columns), which were not of our interest. Then we have eliminated the rows with nan values which could have given some problems later. After these first steps we have noticed that some attributes had more then one value in a single tuple. For example: a film is described by a unique tuple and so it could have more then one director in the attribute column director or/and more then one actor in the attribute column actor. For our scope we needed to separate these values and have only one value for each attribute. There is a function, **df.explode(nameOfColumn)**, that given a column which has lists

[1] https://github.com/AhmedSalahBasha/SchemaMatching

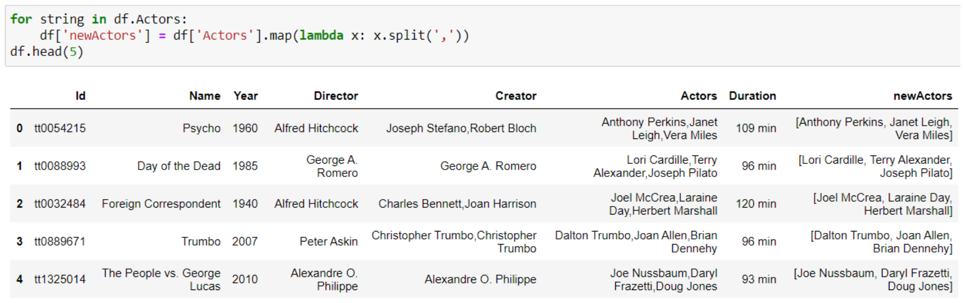
as values, explodes those lists in different rows. It transform each element of a list-like to a row, replicating index values.

As we can see, this function requires the values of the columns to be in a list so, since our values were a string, we had to bring those values in a list. To do so we have used a map function with lambda that given the column transforms the string in a list splitting the value of the string on the , using the **string.split(,)** function. We have saved the lists in new columns. Here we can see how we have applied this step to a mini example (This is only an example were we have created only a new column for attribute "Actors" and we have also dropped some columns of the dataset to render more clear the example):



	Id	Name	Year	Director	Creator	Actors	Duration
0	tt0054215	Psycho	1960	Alfred Hitchcock	Joseph Stefano,Robert Bloch	Anthony Perkins,Janet Leigh,Vera Miles	109 min
1	tt0088993	Day of the Dead	1985	George A. Romero	George A. Romero	Lori Cardille,Terry Alexander,Joseph Pilato	96 min
2	tt0032484	Foreign Correspondent	1940	Alfred Hitchcock	Charles Bennett,Joan Harrison	Joel McCrea,Laraine Day,Herbert Marshall	120 min
3	tt0889671	Trumbo	2007	Peter Askin	Christopher Trumbo,Christopher Trumbo	Dalton Trumbo,Joan Allen,Brian Dennehy	96 min
4	tt1325014	The People vs. George Lucas	2010	Alexandre O. Philippe	Alexandre O. Philippe	Joe Nussbaum,Daryl Frazetti,Doug Jones	93 min

Figure 5.1: Subset of Rotten Tomatoes dataset



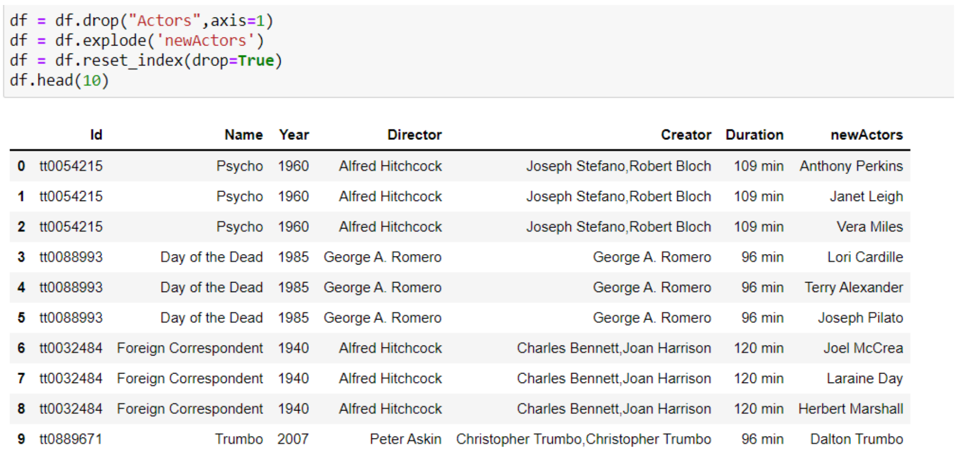
```
for string in df.Actors:
    df['newActors'] = df['Actors'].map(lambda x: x.split(','))
df.head(5)
```

	Id	Name	Year	Director	Creator	Actors	Duration	newActors
0	tt0054215	Psycho	1960	Alfred Hitchcock	Joseph Stefano,Robert Bloch	Anthony Perkins,Janet Leigh,Vera Miles	109 min	[Anthony Perkins, Janet Leigh, Vera Miles]
1	tt0088993	Day of the Dead	1985	George A. Romero	George A. Romero	Lori Cardille,Terry Alexander,Joseph Pilato	96 min	[Lori Cardille, Terry Alexander, Joseph Pilato]
2	tt0032484	Foreign Correspondent	1940	Alfred Hitchcock	Charles Bennett,Joan Harrison	Joel McCrea,Laraine Day,Herbert Marshall	120 min	[Joel McCrea, Laraine Day, Herbert Marshall]
3	tt0889671	Trumbo	2007	Peter Askin	Christopher Trumbo,Christopher Trumbo	Dalton Trumbo,Joan Allen,Brian Dennehy	96 min	[Dalton Trumbo, Joan Allen, Brian Dennehy]
4	tt1325014	The People vs. George Lucas	2010	Alexandre O. Philippe	Alexandre O. Philippe	Joe Nussbaum,Daryl Frazetti,Doug Jones	93 min	[Joe Nussbaum, Daryl Frazetti, Doug Jones]

Figure 5.2: Creation of new column with lists instead of strings

As we can see we had duplicated columns, the ones with the strings and the ones with the lists, so we dropped the first ones which will not be useful to us. Once we dropped those columns we used the explode function on the new columns and then we used the function **df.reset_index(drop=True)** to have our dataframe with correct indexes. We can notice that a film, like "Psycho", had one tuple with the attribute "Actors" that had three actors in one string. Now we have three tuples for "Psycho", one for each actor.

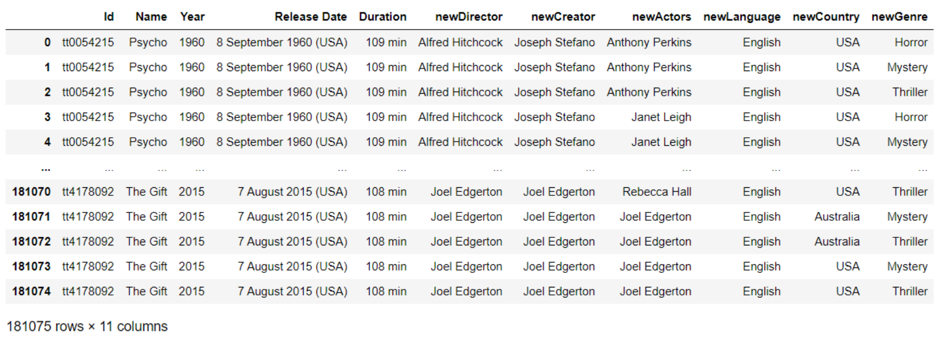
This is the final step and we have obtained the source dataset ready to be passed to the our algorithm. We have saved the final dataset in the sourceComplete.csv file so that we did not have to repeat all the steps to create the dataset all the time. This is how the final dataset look like:



```
df = df.drop("Actors",axis=1)
df = df.explode('newActors')
df = df.reset_index(drop=True)
df.head(10)
```

	Id	Name	Year	Director	Creator	Duration	newActors
0	tt0054215	Psycho	1960	Alfred Hitchcock	Joseph Stefano,Robert Bloch	109 min	Anthony Perkins
1	tt0054215	Psycho	1960	Alfred Hitchcock	Joseph Stefano,Robert Bloch	109 min	Janet Leigh
2	tt0054215	Psycho	1960	Alfred Hitchcock	Joseph Stefano,Robert Bloch	109 min	Vera Miles
3	tt0088993	Day of the Dead	1985	George A. Romero	George A. Romero	96 min	Lori Cardille
4	tt0088993	Day of the Dead	1985	George A. Romero	George A. Romero	96 min	Terry Alexander
5	tt0088993	Day of the Dead	1985	George A. Romero	George A. Romero	96 min	Joseph Pilato
6	tt0032484	Foreign Correspondent	1940	Alfred Hitchcock	Charles Bennett,Joan Harrison	120 min	Joel McCrea
7	tt0032484	Foreign Correspondent	1940	Alfred Hitchcock	Charles Bennett,Joan Harrison	120 min	Laraine Day
8	tt0032484	Foreign Correspondent	1940	Alfred Hitchcock	Charles Bennett,Joan Harrison	120 min	Herbert Marshall
9	tt0889671	Trumbo	2007	Peter Askin	Christopher Trumbo,Christopher Trumbo	96 min	Dalton Trumbo

Figure 5.3: Explode function applied to the created column



	Id	Name	Year	Release Date	Duration	newDirector	newCreator	newActors	newLanguage	newCountry	newGenre
0	tt0054215	Psycho	1960	8 September 1960 (USA)	109 min	Alfred Hitchcock	Joseph Stefano	Anthony Perkins	English	USA	Horror
1	tt0054215	Psycho	1960	8 September 1960 (USA)	109 min	Alfred Hitchcock	Joseph Stefano	Anthony Perkins	English	USA	Mystery
2	tt0054215	Psycho	1960	8 September 1960 (USA)	109 min	Alfred Hitchcock	Joseph Stefano	Anthony Perkins	English	USA	Thriller
3	tt0054215	Psycho	1960	8 September 1960 (USA)	109 min	Alfred Hitchcock	Joseph Stefano	Janet Leigh	English	USA	Horror
4	tt0054215	Psycho	1960	8 September 1960 (USA)	109 min	Alfred Hitchcock	Joseph Stefano	Janet Leigh	English	USA	Mystery
...	...	...	...	...	...	...	...	...	...	...	...
181070	tt4178092	The Gift	2015	7 August 2015 (USA)	108 min	Joel Edgerton	Joel Edgerton	Rebecca Hall	English	USA	Thriller
181071	tt4178092	The Gift	2015	7 August 2015 (USA)	108 min	Joel Edgerton	Joel Edgerton	Joel Edgerton	English	Australia	Mystery
181072	tt4178092	The Gift	2015	7 August 2015 (USA)	108 min	Joel Edgerton	Joel Edgerton	Joel Edgerton	English	Australia	Thriller
181073	tt4178092	The Gift	2015	7 August 2015 (USA)	108 min	Joel Edgerton	Joel Edgerton	Joel Edgerton	English	USA	Mystery
181074	tt4178092	The Gift	2015	7 August 2015 (USA)	108 min	Joel Edgerton	Joel Edgerton	Joel Edgerton	English	USA	Thriller

181075 rows × 11 columns

Figure 5.4: Source Dataset

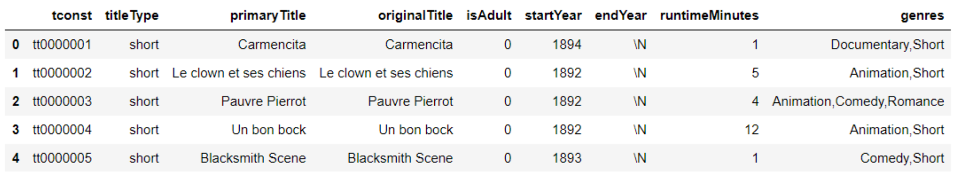
5.1.2 Target Dataset

In the source dataset we had that the dataset was already given as a unique table, instead for the target dataset we have found different dataset linked between each other with primary and foreign keys. These datasets are from the **IMDB** dataset. At this first link there is a description of each dataset: [2] Each dataset is contained in a gzipped, tab-separated-values (TSV) formatted file in the UTF-8 character set. The first line in each file contains headers that describe what is in each column. A \N is used to denote that a particular field is missing or null for that title/name. At this second link [3] there are the downloadable files with the datasets. For our purpose we have created a unique dataset merging these datasets: basics.tsv,crew.tsv,name.tsv,principals.tsv. Now i will detail better how we

[2]https://www.imdb.com/interfaces/
[3]https://datasets.imdbws.com/

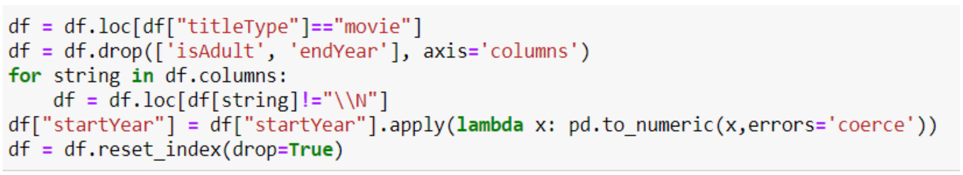
have done it.

Firstly we imported the **basic.tsv** dataset and we selected the tuples where "titleType"=movie. After this we dropped some columns that we will not need, eliminates the \N values and then we have used the **df.apply()** function combined with the lambda and pd.to_numeric functions to be sure that the column "startYear" has all int values (some years were saved as string and not as integer).



	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	Documentary,Short
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	Animation,Short
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4	Animation,Comedy,Romance
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12	Animation,Short
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	Comedy,Short

Figure 5.5: Basic.csv dataset



```
df = df.loc[df["titleType"]=="movie"]
df = df.drop(['isAdult', 'endYear'], axis='columns')
for string in df.columns:
    df = df.loc[df[string]!="\\N"]
df["startYear"] = df["startYear"].apply(lambda x: pd.to_numeric(x,errors='coerce'))
df = df.reset_index(drop=True)
```

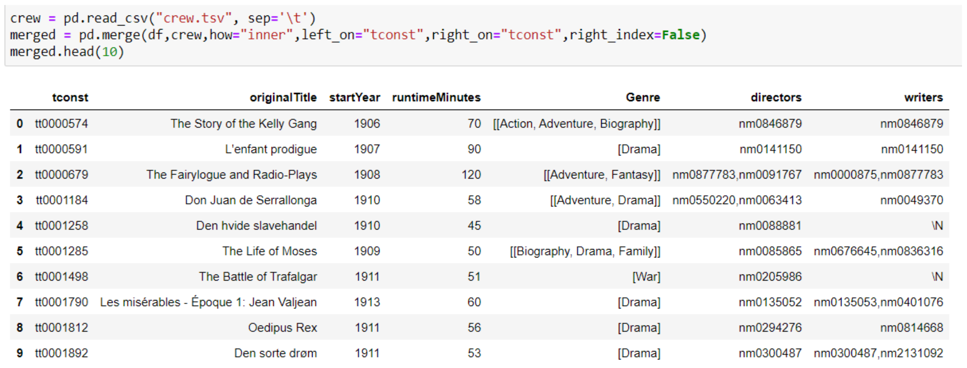
Figure 5.6: Preliminaries steps for the target Dataset

After this, we can see in 5.5 that fro the "genre" column we have the same problem faced in the source dataset, this attribute has more then one value for some tuples, so we will have to use the explode function and before that create the column with the lists obtained from the split string. Before using the explode function that will produce a high number of rows for the dataset, we will do the merge with the other necessary datasets and then explode at the end of all the merges.

As said before now we have to merge the basic.tsv dataset with the other ones. We start with the **crew.tsv** dataset and we merge them using the pd.merge function used like this:

The **merge** perform a join on a specified column of two datasets (the "tconst" column in our case), the result will be a dataset which will have all the columns present in both the dataset and the tuples created from the merge with the "tconst" value.

We then eliminate the rows with \N values and as for the "genre" attribute we create new columns for "director" and "writer", creating columns with lists obtained from the split string. Now we can explode all the columns so we obtain separated values for director and writer that will be fundamen-



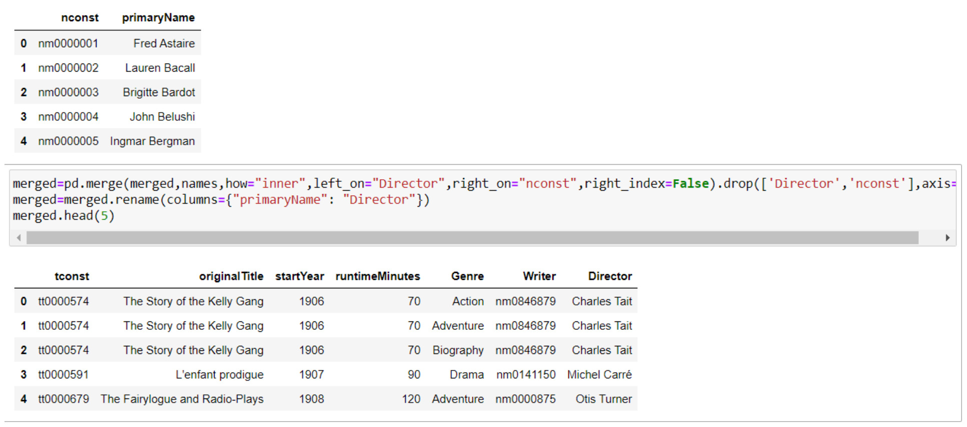
```
crew = pd.read_csv("crew.tsv", sep='\t')
merged = pd.merge(df,crew,how="inner",left_on="tconst",right_on="tconst",right_index=False)
merged.head(10)
```

	tconst	originalTitle	startYear	runtimeMinutes	Genre	directors	writers
0	tt0000574	The Story of the Kelly Gang	1906	70	[[Action, Adventure, Biography]]	nm0846879	nm0846879
1	tt0000591	L'enfant prodigue	1907	90	[Drama]	nm0141150	nm0141150
2	tt0000679	The Fairylogue and Radio-Plays	1908	120	[[Adventure, Fantasy]]	nm0877783,nm0091767	nm0000875,nm0877783
3	tt0001184	Don Juan de Serrallonga	1910	58	[[Adventure, Drama]]	nm0550220,nm0063413	nm0049370
4	tt0001258	Den hvide slavehandel	1910	45	[Drama]	nm0088881	\N
5	tt0001285	The Life of Moses	1909	50	[[Biography, Drama, Family]]	nm0085865	nm0676645,nm0836316
6	tt0001498	The Battle of Trafalgar	1911	51	[War]	nm0205986	\N
7	tt0001790	Les misérables - Époque 1: Jean Valjean	1913	60	[Drama]	nm0135052	nm0135053,nm0401076
8	tt0001812	Oedipus Rex	1911	56	[Drama]	nm0294276	nm0814668
9	tt0001892	Den sorte drøm	1911	53	[Drama]	nm0300487	nm0300487,nm2131092

Figure 5.7: Merge of crew with basic

tal later.

As we can see in 5.7, we don't have the actual names of the directors or actors but we have their codes. We need to substitute these codes with the real names. We can find the names associated to the code in the **name.tsv** column, so with two consecutive merges (one for director and one for writer) we obtain a dataset that has the movies associated with the names of the directors and writers.



	nconst	primaryName
0	nm0000001	Fred Astaire
1	nm0000002	Lauren Bacall
2	nm0000003	Brigitte Bardot
3	nm0000004	John Belushi
4	nm0000005	Ingmar Bergman

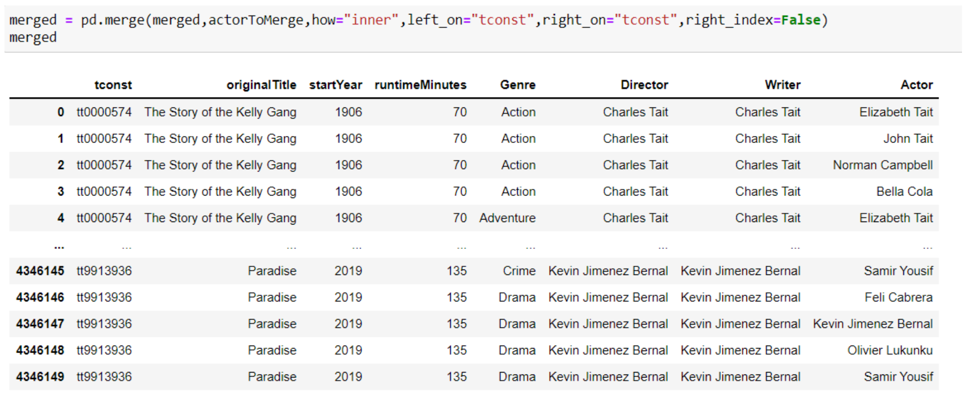
```
merged=pd.merge(merged,names,how="inner",left_on="Director",right_on="nconst",right_index=False).drop(['Director','nconst'],axis=
merged=merged.rename(columns={"primaryName": "Director"})
merged.head(5)
```

	tconst	originalTitle	startYear	runtimeMinutes	Genre	Writer	Director
0	tt0000574	The Story of the Kelly Gang	1906	70	Action	nm0846879	Charles Tait
1	tt0000574	The Story of the Kelly Gang	1906	70	Adventure	nm0846879	Charles Tait
2	tt0000574	The Story of the Kelly Gang	1906	70	Biography	nm0846879	Charles Tait
3	tt0000591	L'enfant prodigue	1907	90	Drama	nm0141150	Michel Carré
4	tt0000679	The Fairylogue and Radio-Plays	1908	120	Adventure	nm0000875	Otis Turner

Figure 5.8: Name dataset and first merge

In 5.8 we can see how the name.tsv dataset, we have already dropped some columns which were not relevant. Then we can see the first merge between our current dataset and the name dataset. We have created a new column called "Director" that instead of the codes has the name of the directors. We still have to change the "Writer" column which is done with a new merge, same as the one just seen.

Now in our dataset we have the films with the directors and writers but without the actors which are saved in another dataset, **principals.tsv**. This dataset has all the person that has worked in a film, so not only actors but all the cast members. So we have initially selected only the tuples that were referring to actors and dropped some columns which were not relevant. Now we have a dataset were the tuples have one code of a film and an actor who worked for that film. Then, as before, the actors are referred with their codes and not with their names. Thus, we have performed a merge between principals and names to obtain a dataset with the film and the name of the actor. Now that we have obtain this dataset we can merge it with our previous dataset to join the tuples with the film, the director and the writer with the tuples with the actors. Our final dataset will be like that:



```
merged = pd.merge(merged,actorToMerge,how="inner",left_on="tconst",right_on="tconst",right_index=False)
merged
```

	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
0	tt0000574	The Story of the Kelly Gang	1906	70	Action	Charles Tait	Charles Tait	Elizabeth Tait
1	tt0000574	The Story of the Kelly Gang	1906	70	Action	Charles Tait	Charles Tait	John Tait
2	tt0000574	The Story of the Kelly Gang	1906	70	Action	Charles Tait	Charles Tait	Norman Campbell
3	tt0000574	The Story of the Kelly Gang	1906	70	Action	Charles Tait	Charles Tait	Bella Cola
4	tt0000574	The Story of the Kelly Gang	1906	70	Adventure	Charles Tait	Charles Tait	Elizabeth Tait
...	...	...	...	...	...	...	...	...
4346145	tt9913936	Paradise	2019	135	Crime	Kevin Jimenez Bernal	Kevin Jimenez Bernal	Samir Yousif
4346146	tt9913936	Paradise	2019	135	Drama	Kevin Jimenez Bernal	Kevin Jimenez Bernal	Feli Cabrera
4346147	tt9913936	Paradise	2019	135	Drama	Kevin Jimenez Bernal	Kevin Jimenez Bernal	Kevin Jimenez Bernal
4346148	tt9913936	Paradise	2019	135	Drama	Kevin Jimenez Bernal	Kevin Jimenez Bernal	Olivier Lukunku
4346149	tt9913936	Paradise	2019	135	Drama	Kevin Jimenez Bernal	Kevin Jimenez Bernal	Samir Yousif

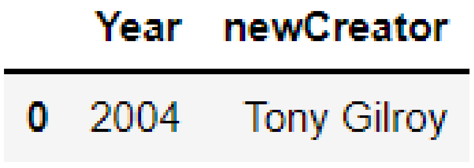
Figure 5.9: Target dataset after the merges

5.2 Experiments

In this chapter we will show the methodology applied to other examples. We have to select two subsets, one from the source and one from the target, in order to have smaller datasets to give as input to the QRE algorithm. Since in the methodology we have introduced an example where in the SELECT-clause we had one attribute, here we will see two examples with two attributes in the SELECT-clause.

5.2.1 First Example

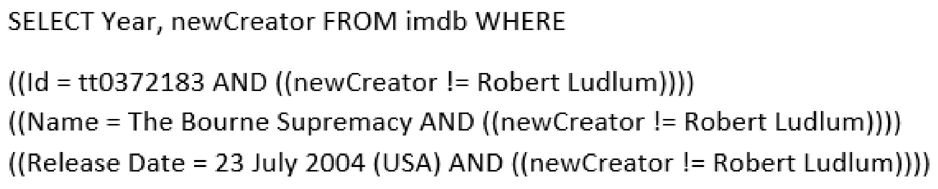
For this example we create the source and the target subsets selecting all the tuples where we have in the attribute actor (”newActors” in the source and Actor in the target) the values ”George Clooney” or ”Matt Damon”. Our result table, what we have obtained as a result from the unknown query, is the following one:



	Year	newCreator
0	2004	Tony Gilroy

Figure 5.10: Result table given as input

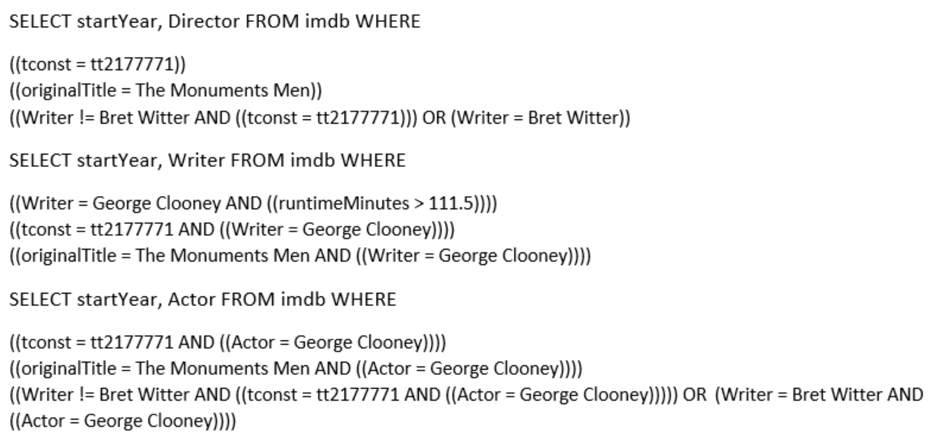
We can see that we have two attributes in this table, so we had two attributes in the SELECT-clause: "Year" and "newCreator". Now we have to apply our methodology to the source and the target with the given result table. The first step will be to find the IEQs from the source. So we will need to encode the source and the result table, then apply QRE to find the positive tuple in the source given the tuples in the result table which will return the IEQs. Here we can see the IEQs obtained from the QRE algorithm:



```
SELECT Year, newCreator FROM imdb WHERE

((Id = tt0372183 AND ((newCreator != Robert Ludlum))))
((Name = The Bourne Supremacy AND ((newCreator != Robert Ludlum))))
((Release Date = 23 July 2004 (USA) AND ((newCreator != Robert Ludlum))))
```

Figure 5.11: IEQs generated from the source

Now that we have the IEQs from the source, we need to find the ones from the target with all the possible combinations of attributes in the SELECT-clause that covers the attribute's values in the result table. Here we have two attribute's values so we could have different numbers of combinations. In our case we are looking for the attributes that covers the values: "2004" and "Tony Gilroy". In the target we will find out that "2004" is covered only by the attribute's column "startYear", instead "Tony Gilroy" is covered by two attribute's columns which are "Director" and "Writer". So we will have two result tables with the combination of the first column with the other two. We will need to call our process to find the IEQs one time for each possible combination. Here we can see the IEQs generated from the target with the different attributes in the SELECT-clauses:



SELECT startYear, Director FROM imdb WHERE

((tconst = tt2177771))
((originalTitle = The Monuments Men))
((Writer != Bret Witter AND ((tconst = tt2177771))) OR (Writer = Bret Witter))

SELECT startYear, Writer FROM imdb WHERE

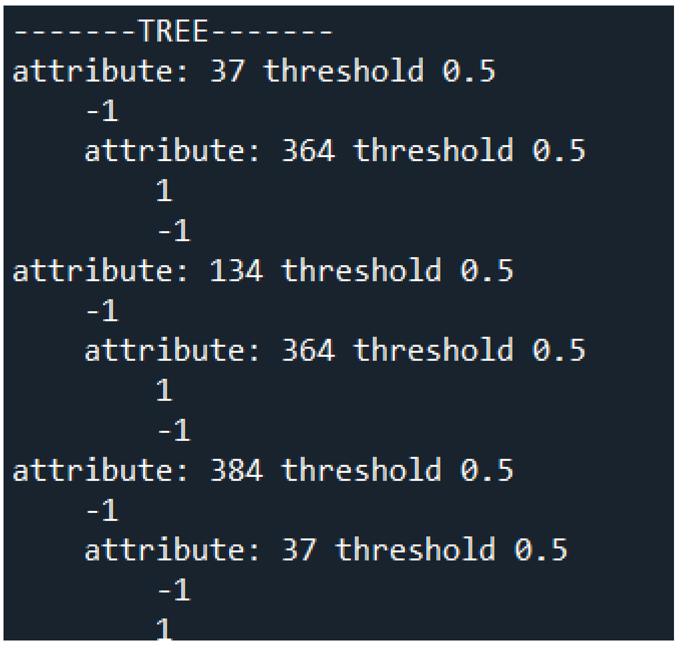
((Writer = George Clooney AND ((runtimeMinutes > 111.5))))
((tconst = tt2177771 AND ((Writer = George Clooney))))
((originalTitle = The Monuments Men AND ((Writer = George Clooney))))

SELECT startYear, Actor FROM imdb WHERE

((tconst = tt2177771 AND ((Actor = George Clooney))))
((originalTitle = The Monuments Men AND ((Actor = George Clooney))))
((Writer != Bret Witter AND ((tconst = tt2177771 AND ((Actor = George Clooney))))) OR (Writer = Bret Witter AND ((Actor = George Clooney))))

Figure 5.12: IEQs generated from the target

In picture 5.12 we have the sets of IEQs obtained from the target, one for each generated result table. We can see that we have a particular case generated from the first result table, the one with Director as attribute. We were looking in the target subset for the tuples with "startYear"=2004 and "Director"="Tony Gilroy" but there aren't tuples that satisfy these conditions so it isn't possible to find IEQs for such case. We have only one set of IEQs obtained from the second result table. Here we can see also the decision tree that generates the queries:

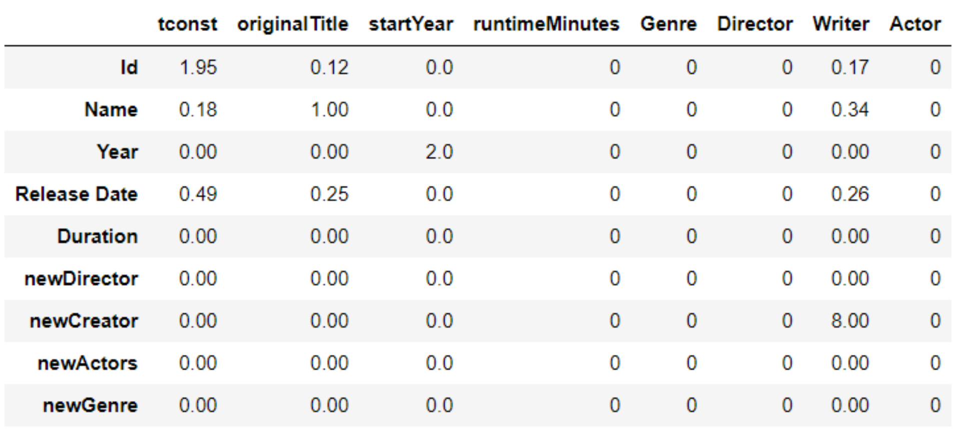


```
-------TREE-------
attribute: 37 threshold 0.5
    -1
    attribute: 364 threshold 0.5
        1
        -1
attribute: 134 threshold 0.5
    -1
    attribute: 364 threshold 0.5
        1
        -1
attribute: 384 threshold 0.5
    -1
    attribute: 37 threshold 0.5
        -1
        1
```

Figure 5.13: Decision trees from the target

Once we have the IEQs from the source and from the target we have to compare them and find which one are more similar. We know that we are looking for the tuples were "Tony Gilroy" appears as a Writer so we know that the comparison between the target queries and the source queries should return as most similar IEQs the set of queries that have as SELECT attribute "Writer". In our case we only have one set of queries so the comparison will of course return it as most similar. This was also the set of queries we were looking for, with "Writer" as attribute.

Finally, we can increase the values in the matrix. Starting from an initialized matrix we obtain the updated matrix in the picture below:



	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
Id	1.95	0.12	0.0	0	0	0	0.17	0
Name	0.18	1.00	0.0	0	0	0	0.34	0
Year	0.00	0.00	2.0	0	0	0	0.00	0
Release Date	0.49	0.25	0.0	0	0	0	0.26	0
Duration	0.00	0.00	0.0	0	0	0	0.00	0
newDirector	0.00	0.00	0.0	0	0	0	0.00	0
newCreator	0.00	0.00	0.0	0	0	0	8.00	0
newActors	0.00	0.00	0.0	0	0	0	0.00	0
newGenre	0.00	0.00	0.0	0	0	0	0.00	0

Figure 5.14: Increased Matrix from the experiment

We can notice an initial mapping between some attributes:

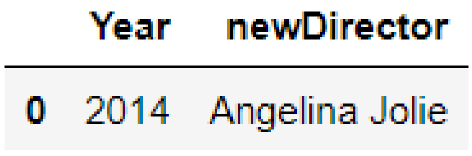
- Id $\longrightarrow$ tconst
- Name $\longrightarrow$ originalTitle
- Year $\longrightarrow$ startyear
- newCreator $\longrightarrow$ Writer

As we said in Chapter 4 this is not sufficient to conclude that those attributes are correctly mapped, we will need to run different examples to generate a truthful matrix.

The program completed in 3.449397087097168 seconds since the queries weren't to much complex and also in the target we only had one set of IEQs to retrieve. We have also compared the compile time using first panda get_dummies() and then one hot encoding. In ten runs with get_dummies() we have obtained an average compile time equal to 3.405. Instead, with one hot encoding we have obtained an average compile time of 8.929. This could be only a case but we can notice how get_dummies() is more optimal than one hot encoding in this example.

5.2.2 Second Example

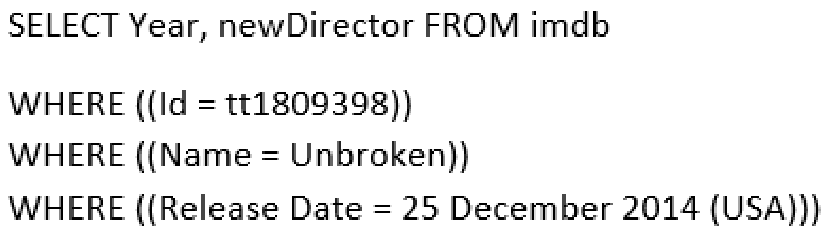
For this example we are going to use the same subsets used in the Methodology chapter, ch 4, the subsets with Angelina Jolie and Ethan Coen. Also this time we will have two attributes in the SELECT-clause but instead of obtaining only one set of IEQs we will have two set of IEQs and so we will be able to look at the comparison between the IEQs. As result table we have:



	Year	newDirector
0	2014	Angelina Jolie

Figure 5.15: Result table of second experiment

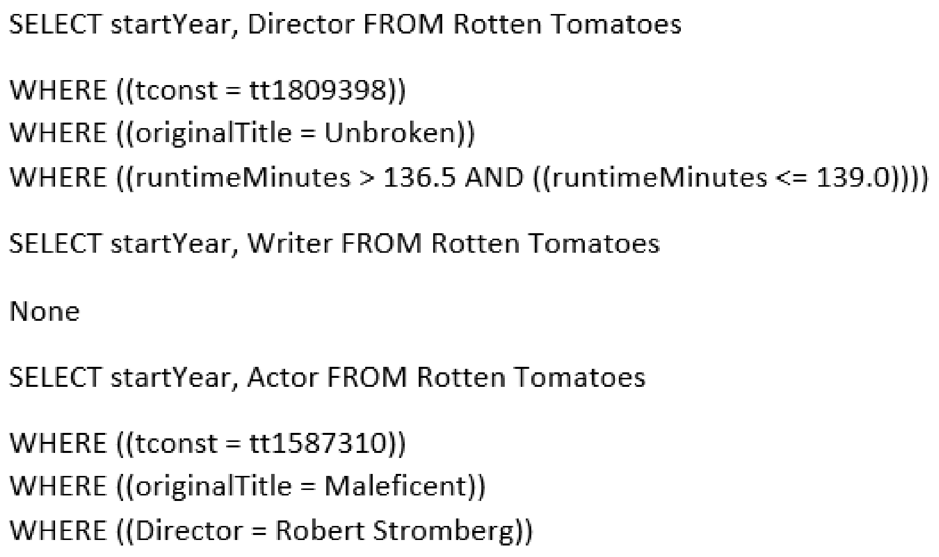
So we are looking for the tuples where the attribute ”Year” has value 2014 and the tuples where the attribute ”newDirector” is equal to ”Angelina Jolie”. We call our function to retrieve the IEQs from the source and we obtain:



```
SELECT Year, newDirector FROM imdb

WHERE ((Id = tt1809398))
WHERE ((Name = Unbroken))
WHERE ((Release Date = 25 December 2014 (USA)))
```

Figure 5.16: IEQs from the source in the second experiment

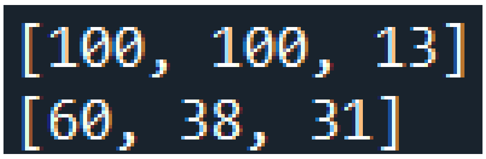
We can now start to generates the possible result tables for the target looking for the covering attribute's column for the values in the original result table. We have that the value ”2014” is covered by the column ”startYear”, instead the attribute ”Angelina Jolie” is covered by three columns (”Director”, ”Writer”, ”Actor”) so we will have three different result table generated by the combination of the ”startYear” column with the other three. Once we have the three result tables, we can call our function passing the target subset and one of the result table. We will have to do this step three times, one for each result table. Every time we will obtain a set of IEQs, if founded. Here we can see the IEQs obtained:



```
SELECT startYear, Director FROM Rotten Tomatoes

WHERE ((tconst = tt1809398))
WHERE ((originalTitle = Unbroken))
WHERE ((runtimeMinutes > 136.5 AND ((runtimeMinutes <= 139.0))))

SELECT startYear, Writer FROM Rotten Tomatoes

None

SELECT startYear, Actor FROM Rotten Tomatoes

WHERE ((tconst = tt1587310))
WHERE ((originalTitle = Maleficent))
WHERE ((Director = Robert Stromberg))
```

Figure 5.17: IEQs from the target in the second experiment

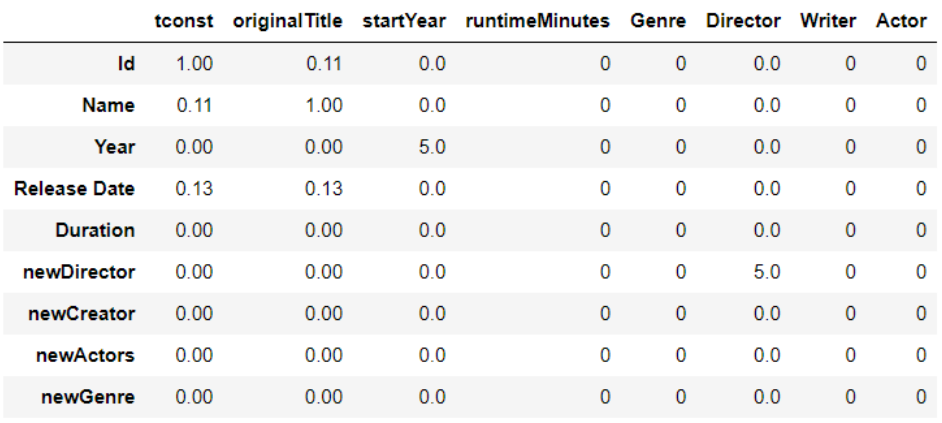
We can notice how the second combination of attributes in the SELECT-clause didn't return any tuples, it means that in the subset there aren't tuples where the attribute "startYear" is equal to "2014" and where the attribute "Writer" is equal to "Angelina Jolie". Unlike the first example we still have two set of IEQs from two different SELECT-clause so we will need to evaluate the WHERE-clause comparing them with the one from the IEQs of the source. We are expecting that the first set of IEQs, the ones that select on the "Director" attribute, will be more similar to the ones of the source. Let's see what we got as result of the comparison:



```
[100, 100, 13]
[60, 38, 31]
```

Figure 5.18: Result of the comparison in the second experiment

It is clear from the results that the first sets of WHERE-clause is more similar to the ones of the source so we select these IEQs. This was what we were expecting so now we have to increment the values of the attributes in the matrix to obtain a partial mapping. Here we have the incremented matrix:



	tconst	originalTitle	startYear	runtimeMinutes	Genre	Director	Writer	Actor
Id	1.00	0.11	0.0	0	0	0.0	0	0
Name	0.11	1.00	0.0	0	0	0.0	0	0
Year	0.00	0.00	5.0	0	0	0.0	0	0
Release Date	0.13	0.13	0.0	0	0	0.0	0	0
Duration	0.00	0.00	0.0	0	0	0.0	0	0
newDirector	0.00	0.00	0.0	0	0	5.0	0	0
newCreator	0.00	0.00	0.0	0	0	0.0	0	0
newActors	0.00	0.00	0.0	0	0	0.0	0	0
newGenre	0.00	0.00	0.0	0	0	0.0	0	0

Figure 5.19: Incremented matrix from the second experiment

Since the WHERE-clauses of the queries both from the target and from the source where simple, with just a few predicates, we have a poor mapping which is not very relevant. We can only hypothesize that the two attributes in the SELECT-clause of the target should map with the two attributes of the source SELECT-clause, the other mappings doesn't have important values to define a secure mapping.

The program completed in 3.9043145179748535 seconds, which is probably a little bit higher of before because we had an extra set of IEQs. Still a short time given the simplicity of the WHERE-clauses.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

In this thesis we addressed the problem of generating a schema mapping from two different data schemas in the same domain with the help of the QRE algorithm, the domain we focalized on was the one of movies. Our work is able to find a mapping not only for the domain we used but also for different domains, given two schemas it can provide the mapping. We have realized a framework that firstly uses a modified QRE algorithm that allows to use also categorical datasets and then, with a function that we created, understands the relationships between the generated IEQs obtained from the QRE and creates the mapping between the schemas. In Chapter 4 we have seen how our framework works and how we have used the string encoding functions and the string comparison functions.

This framework propose an innovative solution in this field since the QRE algorithm was never used, to the best of our knowledge, to implement a schema mapping method. We have exploited its functionality to obtain the IEQs and then implemented the functions to generate the mapping. Furthermore our thesis provide a complete automatic method to obtain a schema mapping, it requires as input only the two datasets and the result table obtained by the unknown query. The user doesn't have to provide any further input which is an important progress with respect to the papers shown in Chapter 3 which needed some user input. With our matrix we have also provided a visual representation of the mapping which clarifies it also to unskilled user.

6.2 Future Works

We have seen that the big data enviroment is constantly changing and growing and so there will be an increasing need for schema mapping methods which allow to unify different schemas with the same domain. The it technologies are expected to keep expanding and so the use of database in different fields. This will bring the need to use schema mapping methods on databases that have a lot of tables with possibly different relationships.

In our studies we have narrowed the problem to the use of two datasets ready to be used by our framework. We have also made some choices necessary to obtain the results but on which we could have probably made a better reasoning. Here we can focalize on some aspects on which we could work in the future to improve our framework:

- Instead of having as input two csv file representing the source and the target datasets, we could have implemented a function that could have taken as input two db file. Initially we have taken as source examples a db with seven different tables and looking at the primary key and foreign key we were able to create different merges depending on the attributes that covered the ones in the result table. We changed this approach because we didn't have a target db with the same domain on which we could have applied the schema mapping method. So in the future we could add at our framework the possibility to take as input csv file or db file and then create different cases according to the type of input. Even with a db file we will need to read the tables, perform the necessary merge and finally obtain the csv file needed by the QRE algorithm.

- Further studies could be made on the evaluation of the where clauses. Here we used the fuzzy string function to compare the attribute's values, we could have used different functions, the ones presented in Chapter 2, and we could have also evaluated the comparison between all the predicate (the attribute's name and the attribute's value) and not only the attribute's value. In addition to that we could have also evaluated differently the results obtained by the similarity between numerical values. In the different possible cases presented in Chapter 4 we had a large number of different intervals and we could find a possible better way to evaluate those intervals also looking at the type of number we have, for example if they are similar in number of digits or not.

- As we saw in Chapter 2 we have different functions that perform string encoding. We have used pd.get_dummies() but we could have used something else, further studies could prove that a different function may bring some optimization or can create smaller datasets which would be more easier to be used by the QRE algorithm.

- The QRE algorithm on which we have implemented our framework used as decision tree the one from sklearn, which generated the split by looking at the Gini index values. There are different types of decision trees that work differently, it is possible that another type of decision tree could be more efficient for our work since we are dealing with possibly very large datasets.

Bibliography

[1] Bogdan Alexe, Balder TEN Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. 36(4), 2011.

[2] P. Atzeni, Luigi Bellomarini, Paolo Papotti, and Riccardo Torlone. Meta-mappings for schema mapping reuse. *Proc. VLDB Endow.*, 12:557–569, 2019.

[3] Zohra Bellahsene, Angela Bonifati, Fabien Duchateau, and Yannis Velegrakis. *On Evaluating Schema Matching and Mapping*, pages 253–291. 12 2011.

[4] Angela Bonifati, Ugo Comignani, Emmanuel Coquery, and Romuald Thion. Interactive mapping specification with exemplar tuples. pages 667–682. ACM, 2017.

[5] B. T. Cate, Phokion G. Kolaitis, and W. Tan. Schema mappings and data examples. In *EDBT '13*, 2013.

[6] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment*, 5, 03 2012.

[7] Ronald Fagin, Laura Haas, Mauricio Hernndez, Rene Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. pages 198–236, 01 2009.

[8] Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. 57(2), 2010.

[9] Jorge Gracia, Vanessa Lopez, Mathieu d'Aquin, Marta Sabou, Enrico Motta, and Eduardo Mena. Solving semantic ambiguity to improve semantic web based ontology matching. volume 304, 2007.

[10] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chang. Accessing the deep web: A survey. *Commun. ACM*, 50:94–101, 05 2007.

[11] Xian Li, Xin Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? *Proceedings of the VLDB Endowment*, 6, 03 2015.

[12] H. V. Jagadish Li Qian, Michael J. Cafarella. Sample-driven schema mapping. In *In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD 12*, pages 73–84. ACM, 2012.

[13] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (Luna) Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. In *CIDR*, 2007.

[14] Srinivasan Parthasarathy Quoc Trung Tran, Chee-Yong Chan. Query reverse engineering. *The VLDB journal*, 23:721–746, 2014.

[15] Clare Southerton. *Datafication*, pages 1–4. Springer International Publishing, Cham, 2020.

[16] Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Schema mappings and data examples. Association for Computing Machinery, 2013.

[17] Quoc Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23, 10 2014.

[18] Divesh Srivastava Xin Luna Dong. *Big Data Integration.* University of Waterloo, 2015.